

Trabajo Fin de Grado

Grado en Ingeniería de las Telecomunicaciones

Análisis, Documentación e Implementación del analizador de red nftables en sistemas Linux

Autor: Francisco Javier Rodríguez López

Tutor: Javier Muñoz Calle

Dpto. Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021



Trabajo Fin de Grado
Grado en Ingeniería de las Telecomunicaciones

Análisis, Documentación e Implementación del analizador de red nftables en sistemas Linux

Autor:
Francisco Javier Rodríguez López

Tutor:
Profesor titular
Javier Muñoz Calle

Dpto. de Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo Fin de Grado: Análisis, Documentación e Implementación del analizador de red nftables en sistemas Linux

Autor: Francisco Javier Rodríguez López

Tutor: Javier Muñoz Calle

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

A mi pareja

A mis abuelos

A mis amigos

A mis maestros

Agradecimientos

En este punto, me gustaría agradecer a todas aquellas personas que me han apoyado y animado a cumplir esta meta de ser ingeniero en telecomunicaciones.

Principalmente a mi familia, a mis padres y a mi hermana por estar ahí siempre que las cosas no salían como estaban previstas, por apoyarme y animarme a seguir adelante, por aguantar las noches programando y estudiando, por enseñarme los principios que hoy marcan mi día a día y por brindarme la posibilidad de estudiar esta carrera. Me faltan días para agradecerlos todo lo que hacéis por mí.

A mi pareja, sabes que, sin tí , esto no hubiera sido posible y que al igual que has estado para mí como pilar fundamental en mis días, yo estaré ahí siempre para tí.

A mis telequitos, sin ustedes la salud mental la habría perdido hace mucho, gracias porque vuestros son los mejores recuerdos que tengo guardado en ese edificio de paredes rojas, por el apoyo y las regañinas. Porque todavía nos queda mucho por disfrutar y ahora es cuando comienza otra etapa.

A mis grupo de los salesianos(los papitos), porque hacéis que los meses sean un poco menos malos y por el apoyo que habéis dado y aunque el trabajo nos lleve a ciudades distintas, siempre nos quedará discord, las quedadas y el LoL.

A mi tutor Javier Muñoz Calle y a Pablo Neira, por guiarme por este proyecto tan complicado y aportar para que pudiera salir adelante.

Y por último, pero no por ello menos importante, a dos personas que no han podido ver como se llegaba al final de este largo y duro camino, mis logros son también los vuestros. Gracias por que me disteis el regalo más bonito que se podía dar, el pasar tiempo a vuestro lado.

Francisco Javier Rodríguez López

Sevilla, 2021

Resumen

Actualmente, la seguridad de la información y de nuestros equipos, ha ido cobrando más y más importancia.

Ante este aumento en la concienciación sobre la importancia de nuestros datos surgen un sin fin de herramientas que nos permiten bastionar y securizar los sistemas.

Este proyecto nace del desconocimiento de una nueva herramienta como es nftables, y busca aclarar aspectos acerca del comportamiento de esta, así como exemplificar escenarios donde se pueda ver con detalle el uso de sus diferentes opciones, así como el comportamiento real de la misma (ya que en muchos casos dista bastante de la información imprecisa e inexacta que se encuentra en la red).

También se busca dar respuesta no solo al funcionamiento e implementación de los módulos de nftables, sino estudiar su comportamiento en conjunción con los módulos NAT y Conntrack.

El enfoque de este proyecto es práctico, por lo que cada explicación teórica, usará un escenario para exemplificar y mostrar el uso de la misma.

El proyecto se abordará con un enfoque ascendente, y se comenzará explicando las opciones básicas del mismo, así como su configuración, y se irá avanzando por los distintos temas, hasta finalizar abordando el uso de los módulos de esta.

El único sistema completamente seguro es aquel que está apagado, encerrado en un bloque de cemento y sellado en una habitación rodeada de alambradas y guardias armados.

Gene Spafford

Abstract

Currently, the security of information and our equipment has been gaining more and more importance.

Faced with this increase in awareness of the importance of our data, a myriad of tools emerge that allow us to manage and secure systems.

This project was born from the ignorance of a new tool such as nftables, and seeks to clarify aspects about its behavior, as well as exemplify scenarios where the use of its different options can be seen in detail, as well as its real behavior (already which in many cases is far from the imprecise and inaccurate information found on the network).

It also seeks to respond not only to the operation and implementation of the nftables modules, but to study their behavior in conjunction with the NAT and Conntrack modules.

The approach of this project is practical, so each theoretical explanation will use a scenario to exemplify and show the use of it.

The project will be approached with a bottom-up approach, and it will begin by explaining the basic options of the same, as well as its configuration, and will progress through the different topics, until finishing addressing the use of the modules of this

-translation by google-

Índice

Agradecimientos	ix
Resumen	xi
Abstract	xiii
Índice	xv
Índice de Tablas	xix
Índice de Figuras	xxi
Nomenclatura	xxix
1 Introducción	1
1.1 <i>Motivación del Proyecto</i>	1
1.2 <i>Estructura del Proyecto</i>	1
1.3 <i>Alcance y Objetivos</i>	2
2 Estado del Arte	3
2.1 <i>Estado actual de la herramienta</i>	3
2.2 <i>Estudio de Mercado</i>	4
3 Nftables Introducción	7
3.1 <i>Instalación Nftables</i>	7
3.2 <i>Construcción básica nftables</i>	10
3.2.1 Tablas	12
3.2.2 Cadenas y Hooks	12
3.2.3 Reglas básicas	21
4 Flujo de Paquetes (Packet Flow)	33
4.1 <i>Diagrama Flujo de paquetes</i>	33
4.1.1 Camino Bridge	34
4.1.2 Camino ARP	34
4.1.3 Camino Ip	35
4.2 <i>Escenario pruebas flow paquetes.</i>	36
4.2.1 Ejemplo flow paquetes.	36
4.3 <i>Escenario prueba flow interfaz lo</i>	44
4.3.1 Ejemplo flow interfaz lo.	44
5 Managers	51
5.1 <i>Firewalld</i>	51
5.1.1 Escenario de Pruebas	52
6 Seguimiento de Conexiones(Conntrack)	61
6.1 <i>Máquina de estados.</i>	61

6.1.1	Ejemplo de clasificación de estados	64
6.2	<i>Connection tracking Linux</i>	66
6.2.1	Entrada conntrack TCP	69
6.2.2	Entrada conntrack UDP	71
6.2.3	Entrada conntrack ICMP	72
6.2.4	Ejemplo de Seguimiento de Conexiones	75
6.3	<i>TCP Conntrack</i>	79
6.3.1	Ejemplo conntrack TCP.	80
6.3.2	Ejemplo conntrack TCP invalid.	90
6.4	<i>UDP Conntrack</i>	97
6.4.1	Ejemplo UDP conntrack.	98
6.5	<i>ICMP Conntrack</i>	102
6.5.1	Ejemplo ICMP conntrack.	102
6.5.2	Ejemplo ICMP conntrack reject.	106
6.6	<i>Parámetros Conntrack</i>	111
6.7	<i>Helpers</i>	115
6.8	<i>Pruebas escenario con módulo ct (Connection Tracking)</i>	129
6.8.1	Ejemplo mark	129
6.8.2	Ejemplo state	132
7	Traducción de direcciones (NAT)	137
7.1	<i>Snat y Masquerade</i>	137
7.1.1	Ejemplo Snat	137
7.1.2	Ejemplo Masquerade	144
7.2	<i>Dnat y Redirect</i>	147
7.2.1	Ejemplo Redirect	148
7.3	<i>El módulo conntrack en NAT</i>	151
7.3.1	Stateless NAT	151
7.3.2	Stateful NAT	156
7.4	<i>Ejemplo Redirect en Prerouting</i>	163
7.5	<i>Ejemplo Redirect en Output</i>	170
7.6	<i>Ejemplo NAT Asimétrico</i>	176
8	Filtrado de paquetes	185
8.1	<i>Conjuntos Dinámicos</i>	185
8.1.1	Ejemplo de port-knocking	186
8.2	<i>Mapas Dinámicos</i>	192
8.2.1	Ejemplo mapa direcciones Ip	193
8.2.2	Ejemplo mapa protocolos	196
8.3	<i>Medidas de tráfico</i>	201
8.3.1	Limitación de ratio	201
8.3.2	Quotas	203
8.3.3	Contadores nombrados	206
8.4	<i>Ejemplo lista negra</i>	208
9	Conclusiones	213
9.1	<i>Objetivos conseguidos</i>	213
9.2	<i>Futuras líneas de investigación</i>	215
Anexo A - Tecnologías Usadas Y herramientas		217
<i>VMWare</i>		217
<i>Wireshark y TCPdump</i>		221
<i>Hping3, Curl</i>		222
Anexo B - Instalación Nftables		225
<i>Instalación de dependencias</i>		225
<i>Instalación de Nftables</i>		228

Anexo C - Instalación Conntrack-Tools	231
<i>Instalación dependencias de Conntrack-tools</i>	231
<i>Instalación de Conntrack-tools</i>	232
Anexo D - Instalación y Configuración Firewalld	233
<i>Instalación de dependencias Firewalld</i>	233
<i>Instalación de Firewalld</i>	235
<i>Configuración de Firewalld</i>	236
Anexo E - Escenario 2 Equipos	243
<i>Configuración del equipo PC1</i>	244
<i>Configuración del equipo Encaminador</i>	246
Anexo F - Escenario 3 Equipos	249
<i>Configuración del equipo PC1</i>	250
<i>Configuración del equipo PC2</i>	252
<i>Configuración del equipo Encaminador</i>	255
<i>Caso específico: Escenario NAT Asimétrico</i>	258
Configuración del equipo Router NAT	259
Anexo G - CheatSheet Nftables	261
<i>Línea de comandos CheatSheet</i>	261
<i>Módulos CheatSheet</i>	263
<i>Conntrack-Tools CheatSheet</i>	269
Anexo H - Repaso Protocolos (TCP,UDP,ICMP)	271
<i>Protocolo TCP (Transmission Control Protocol)</i>	271
Cabeza TCP	271
Establecimiento de la conexión TCP	272
Cierre de la conexión TCP	273
<i>Protocolo UDP (User Datagram Protocol)</i>	275
Cabeza UDP	275
<i>Protocolo ICMP (Internet Control Message Protocol)</i>	276
Cabeza ICMP	276
Anexo I - Concepto Conexión	281
<i>Protocolo orientado a conexión.</i>	281
<i>Socket de conexión.</i>	281
<i>Conexión en conntrack</i>	282
Referencias	283

ÍNDICE DE TABLAS

Tabla 1 - Familias nftables	11
Tabla 2 - Tipos de cadenas	14
Tabla 3 - Enganches Netfilter	15
Tabla 4 - Prioridades de las cadenas	15
Tabla 5 - Acciones terminales reglas	22
Tabla 6 - Acciones no terminales	28
Tabla 7 - Zonas Firewalld	52
Tabla 8 - Puertos y servicios libvirt Firewalld	58
Tabla 9 - Estados conntrack	62
Tabla 10 - Campos de la entrada conntrack TCP	70
Tabla 11 - Campos de la entrada conntrack UDP	72
Tabla 12 - Campos de la entrada conntrack ICMP	74
Tabla 13 - Estados entradas conntrack	74
Tabla 14 - Valores por defecto estados TCP	79
Tabla 15 - Resumen paquetes TCP invalid	97
Tabla 16 - Variables configuración conntrack	113
Tabla 17 - Campos de la entrada conntrack expectation	125
Tabla 18 - Protocolos con helper en conntrack	129
Tabla 19 - Resumen escenario 2 equipos	243
Tabla 20 - Resumen escenario 3 equipos	249
Tabla 21 - Resumen escenario NAT asimétrico	258
Tabla 22 - CheatSheet comando table	261
Tabla 23 - CheatSheet comando chain	262
Tabla 24 - CheatSheet comando rule	262
Tabla 25 - CheatSheet módulo Ip	263
Tabla 26 - CheatSheet módulo TCP	264
Tabla 27 - CheatSheet módulo UDP	265
Tabla 28 - CheatSheet módulo ICMP	265
Tabla 29 - CheatSheet módulo ether	266
Tabla 30 - CheatSheet módulo ARP	266
Tabla 31 - CheatSheet módulo ct	267
Tabla 32 - CheatSheet módulo meta	269

Tabla 33 - CheatSheet Conntrack-Tools	269
Tabla 34 - Cabecera TCP	272
Tabla 35 - Cabecera UDP	275
Tabla 36 - Tipo mensajes ICMP	279

ÍNDICE DE FIGURAS

Figura 2.1- Últimas versiones Herramientas Netfilter	3
Figura 2.2 - (Fortigate 60F)	5
Figura 3.1 - Versión del kernel	7
Figura 3.2 - Comprobación instalación	8
Figura 3.3 - Módulos cargados en el kernel	9
Figura 3.4 - Listar tablas	12
Figura 3.5 - Ejemplo 1 cadena usuario	17
Figura 3.6 - Resultado ejemplo 1 cadena usuario	18
Figura 3.7 - Ejemplo 2 cadena usuario	19
Figura 3.8 - Resultado Ejemplo 2 cadena usuario	20
Figura 3.9 - Iptables translate	22
Figura 3.10 - Ejemplo reject	23
Figura 3.11 - Resultado ejemplo reject	24
Figura 3.12 - Ejemplo return, accept y drop	26
Figura 3.13 - Resultado ejemplo return, accept y drop	27
Figura 3.14 - Ejemplo acción log	29
Figura 3.15 - Resultado del ejemplo log	30
Figura 3.16 - Ejemplo posición counter	31
Figura 3.17 - Resultado ejemplo posición counter	32
Figura 4.1 - Flow de paquetes de Nftables	33
Figura 4.2 - Bridge Flow	34
Figura 4.3 - ARP Flow	35
Figura 4.4 - Ip Flow	35
Figura 4.5 - Armazón hooks escenario flow	37
Figura 4.6 - Reglas escenario flow	39
Figura 4.7 - Prueba ICMP PC1 escenario flow	39
Figura 4.8 - Nftrace ICMP local, entrada	40
Figura 4.9 - Nftrace ICMP local Salida	40
Figura 4.10 - Logs del ICMP local al pasar por los hooks	41
Figura 4.11 - Resumen camino ping a Encaminador	41

Figura 4.12 - Nftrace ICMP externo entrada	42
Figura 4.13 - Logs del ICMP externo al pasar por los hooks	43
Figura 4.14 - Resumen camino ping a DNS Google	43
Figura 4.15 - Reglas escenario flow interfaz lo	45
Figura 4.16 - Nftrace ICMP req interfaz lo	45
Figura 4.17 - Nftrace ICMP rep interfaz lo	46
Figura 4.18 - Regla ingress interfaz lo	47
Figura 4.19 - Captura Wireshark interfaz lo	47
Figura 4.20 - Registro logs ICMP interfaz lo	48
Figura 4.21 - Resumen camino ping lo	48
Figura 5.1 - Estado servicio Firewalld	53
Figura 5.2 - Zonas definidas básicas	54
Figura 5.3 - Configuración zona Internal	55
Figura 5.4 - Arranque del servidor Apache	55
Figura 5.5 - Flujo conexión Apache (Internal)	56
Figura 5.6 - Página test Apache(Internal)	56
Figura 5.7 - Redirección de puertos (Internal)	57
Figura 5.8 - Reglas libvirt Firewalld	58
Figura 5.9 - Reglas zona Internal Firewalld	59
Figura 5.10 - Reglas forward Firewalld	59
Figura 6.1 - Puntos de clasificación de los paquetes con conntrack	63
Figura 6.2 - Estructura de cadenas y reglas del ejemplo de clasificación	65
Figura 6.3 - Contadores de las reglas del ejemplo de clasificación de estados	65
Figura 6.4 - Creación de las entradas conntrack	68
Figura 6.5 - Entrada conntrack TCP extendido	69
Figura 6.6 - Entrada conntrack UDP extendido	71
Figura 6.7 - Entrada conntrack ICMP extendido	72
Figura 6.8 - Estados entradas conntrack modo monitor	74
Figura 6.9 - Mostrar entradas conntrack	75
Figura 6.10 - Añadir regla counter en prerouting	76
Figura 6.11 - Reglas NAT escenario 2 equipos	76
Figura 6.12 - Tabla de conexiones con tuplas	76
Figura 6.13 - Conexiones con servidores web	77
Figura 6.14 - Entrada conexión con servidor trajano.us.es	77
Figura 6.15 - Flujo de conexión con servidor trajano	77
Figura 6.16 - Eliminación de tuplas	78
Figura 6.17 - Mostrar el número de entradas que tiene conntrack	78
Figura 6.18 - Estados TCP definidos en el código de Netfilter	79
Figura 6.19 - Estructura TCP conntrack	81

Figura 6.20 - Camino paquete TCP SYN	82
Figura 6.21 - Camino paquete TCP SYN-ACK	83
Figura 6.22 - Camino paquete TCP ACK	83
Figura 6.23 - Esquema estados conntrack TCP handshake	84
Figura 6.24 - Entradas conntrack paquete TCP	85
Figura 6.25 - Flow ejemplo TCP handshake conntrack	85
Figura 6.26 - Camino paquete TCP FIN-ACK PC1	85
Figura 6.27 - Camino paquete TCP FIN-ACK Encaminador	86
Figura 6.28 - Camino paquete TCP ACK cierre conexión	87
Figura 6.29 - Entradas conntrack cierre conexión TCP	87
Figura 6.30 - Esquema estados conntrack TCP close connection	88
Figura 6.31 - Flow ejemplo TCP cierre conexión conntrack	89
Figura 6.32 - Envejecimiento entrada conntrack	89
Figura 6.33 - Entradas conntrack del cierre de conexión del Encaminador	89
Figura 6.34 - Camino TCP PSH	92
Figura 6.35 - Camino TCP URG	93
Figura 6.36 - Camino TCP RST	93
Figura 6.37 - Camino TCP FIN	94
Figura 6.38 - Camino TCP ACK	94
Figura 6.39 - Camino TCP SYN	95
Figura 6.40 - Camino TCP SYN-ACK	96
Figura 6.41 - Entrada conntrack TCP-SYN	96
Figura 6.42 - Entrada conntrack TCP-ACK	97
Figura 6.43 - Camino de un paquete UDP	99
Figura 6.44 - Captura de paquete UDP	99
Figura 6.45 - Comando nslookup para consultar DNS	100
Figura 6.46 - Entradas conntrack paquete UDP	100
Figura 6.47 - Esquema estados conntrack UDP	101
Figura 6.48 - Camino paquetes conntrack UDP	101
Figura 6.49 - Captura echo-request y echo-reply ICMP conntrack	104
Figura 6.50 - Camino paquete echo-request	104
Figura 6.51 - Entradas conntrack creadas por comunicación ICMP	105
Figura 6.52 - Envío de paquete echo-reply	106
Figura 6.53 - Reglas reject de ICMP	107
Figura 6.54 - Paquete reject con estado related	108
Figura 6.55 - Respuesta de la regla reject	108
Figura 6.56 - Camino paquete echo-request ejemplo reject	109
Figura 6.57 - Camino paquete echo-reply generado por reject	109
Figura 6.58 - Evento conntrack del request del ejemplo reject	110

Figura 6.59 - Entrada conntrack del request del ejemplo reject	110
Figura 6.60 - Aumento del contador related de ICMP	111
Figura 6.61 - Valores parámetros conntrack en Encaminador	114
Figura 6.62 - Reglas para ct ftp-1	118
Figura 6.63 - Reglas para ct ftp-2	118
Figura 6.64 - Conexión FTP desde PC1	119
Figura 6.65 - Paquete TCP SYN flujo control FTP	120
Figura 6.66 - Paquete TCP SYN-ACK flujo control FTP	121
Figura 6.67 - Paquete TCP ACK flujo control FTP	122
Figura 6.68 - Entradas conntrack conexión flujo de control FTP.	122
Figura 6.69 - Campos de una entrada expectation conntrack	123
Figura 6.70 - Paquete TCP SYN flujo datos FTP	125
Figura 6.71 - Paquete TCP SYN-ACK flujo datos FTP	126
Figura 6.72 - Paquete TCP ACK flujo datos FTP	127
Figura 6.73 - Entradas conntrack ejemplo FTP	127
Figura 6.74 - Entrada conntrack en la tabla expect, ejemplo FTP	128
Figura 6.75 - Reglas ejemplo ct mark	131
Figura 6.76 - Ping PC1 ejemplo ct mark	131
Figura 6.77 - Flujo de conexión ejemplo ct mark 0x2	132
Figura 6.78 - Ruleset ejemplo ct state	133
Figura 6.79 - Flujo conexión servidor Apache	134
Figura 6.80 - Captura Tráfico conexión PC1 servidor Apache de Encaminador	134
Figura 6.81 - Petición a Google desde PC1	135
Figura 6.82 - Captura tráfico sin resolución DNS PC1	135
Figura 6.83 - Aumento de paquetes drop en la cadena forward	135
Figura 7.1 - Intento conexión trajano desde PC1 sin NAT	138
Figura 7.2 - Petición DNS lanzada por PC1 sin NAT en Encaminador	138
Figura 7.3 - Petición DNS de PC1 sale por Encaminador sin NAT	139
Figura 7.4 - Armazón ejemplo Snat y Dnat	140
Figura 7.5 - Regla Snat en Encaminador	140
Figura 7.6 - Paquete Req y Res DNS	141
Figura 7.7 - Paquete Req y Res DNS con Snat aplicado en Encaminador	141
Figura 7.8 - Conexión con Servidor web Trajano	142
Figura 7.9 - Captura ens38 Encaminador conexión Trajano con Snat completo	142
Figura 7.10 - Captura ens33 Encaminador conexión Trajano Snat completo	143
Figura 7.11 - Reglas y contadores Snat de Encaminador	143
Figura 7.12 - Captura ens38 Encaminador usando Masquerade	145
Figura 7.13 - Captura ens33 Encaminador usando Masquerade	145
Figura 7.14 - Nueva dirección Ip ens33 con Masquerade	146

Figura 7.15 - Captura ens38 Encaminador con Masquerade nueva IP	146
Figura 7.16 - Captura ens33 Encaminador con Masquerade nueva IP	147
Figura 7.17 - Reglas y contadores Masquerade de Encaminador	147
Figura 7.18 - Redirect de servidor Apache al de la máquina Encaminador	149
Figura 7.19 - Servidor Apache servido a PC2 con redirect en Encaminador	149
Figura 7.20 - Petición Apache de Encaminador puerto 80 con redirect de puerto	150
Figura 7.21 - Petición Apache de Encaminador puerto 452 con redirect de puerto	150
Figura 7.22 - Página web servida desde el puerto 452 de Encaminador con Redirect	151
Figura 7.23 - Camino Icmp-request Stateless NAT	154
Figura 7.24 - Camino Icmp-reply Stateless NAT	155
Figura 7.25 - Reglas y contadores Stateless NAT	156
Figura 7.26 - Carencia de tablas conntrack Stateless NAT	156
Figura 7.27 - Camino masquerade Stateful NAT	158
Figura 7.28 - Entrada conntrack NAT	159
Figura 7.29 - Trace para ver el cambio inverso de NAT	160
Figura 7.30 - Contadores para comprobar cambio inverso de NAT	160
Figura 7.31 - Argumento notrack y carencia de cadena nat	161
Figura 7.32 - Tras quitar argumento notrack, cadena prerouting tipo nat	162
Figura 7.33 - Camino ping PC1 redirect en prerouting	165
Figura 7.34 - Captura PC1 ping redirect en prerouting	166
Figura 7.35 - Entrada conntrack NAT ICMP redirect en prerouting	166
Figura 7.36 - Monitor conntrack NAT ICMP redirect en prerouting	166
Figura 7.37 - Camino TCP redirect en prerouting	167
Figura 7.38 - Monitor conntrack TCP redirect en prerouting	168
Figura 7.39 - Camino UDP redirect en prerouting	168
Figura 7.40 - Monitor conntrack UDP redirect en prerouting	169
Figura 7.41 - Actualizada entrada UDP redirect en prerouting	169
Figura 7.42 - Estructura nftables Redirect en prerouting	170
Figura 7.43 - Estructura redirect en output	172
Figura 7.44 - Camino ICMP externo redirect en output	173
Figura 7.45 - Camino ICMP interno redirect en output	174
Figura 7.46 - Camino completo ICMP redirect en output	175
Figura 7.47 - Captura tráfico interfaz “lo”, redirect en output	176
Figura 7.48 - Entradas NAT conntrack , redirect en output	176
Figura 7.49 - Estructura nftables Encaminador, escenario asimétrico	179
Figura 7.50 - Regla Router NAT escenario asimétrico	180
Figura 7.51 - Captura interfaz ens37 PC1 escenario asimétrico	181
Figura 7.52 - Camino echo-request Encaminador en escenario asimétrico	181
Figura 7.53 - Captura interfaz ens39 Encaminador en escenario asimétrico	182

Figura 7.54 - Captura ens37 Router NAT escenario asimétrico	182
Figura 7.55 - Captura ping PC1 escenario asimétrico	182
Figura 7.56 - Captura ens37 PC1 echo reply escenario asimétrico	183
Figura 7.57 - Entradas conntrack Encaminador escenario asimétrico	183
Figura 7.58 - Entradas conntrack equipo Router NAT escenario asimétrico	183
Figura 8.1 - Set dinámico	185
Figura 8.2 - Secuencia de port-knocking	188
Figura 8.3 - Primer intento de acceso al servidor web	189
Figura 8.4 - Primera secuencia del port knocking	189
Figura 8.5 - Segunda secuencia del port-knocking	190
Figura 8.6 - Petición al servidor aceptada después del port-knocking	190
Figura 8.7 - Set's actualizados dinámicamente	191
Figura 8.8 - Resumen camino paquetes ejemplo port-knocking	192
Figura 8.9 - Ping PC1 ejemplo mapa	194
Figura 8.10 - Camino paquetes ejemplo mapa	195
Figura 8.11 - Camino ejemplo mapa direcciones Ip's	196
Figura 8.12 - Estructura reglas ejemplo mapa protocolos	198
Figura 8.13 - Cadenas del mapa después de procesar tráfico	199
Figura 8.14 - Camino paquete ICMP ejemplo mapa protocolos	199
Figura 8.15 - Camino de paquete UDP ejemplo mapa protocolos	200
Figura 8.16 - Camino de paquete TCP ejemplo mapa protocolos	200
Figura 8.17 - Camino ejemplo mapa direcciones protocolos	201
Figura 8.18 - Reglas limit rate	202
Figura 8.19 - Ejemplo quota	204
Figura 8.20 - Quota cumplida y paquetes con acción drop	204
Figura 8.21 - Paquetes antes de cumplir la quota	205
Figura 8.22 - Paquetes después de cumplir la quota	205
Figura 8.23 - Estructura contador nombrado y reglas	207
Figura 8.24 - Prueba ejemplo contador nombrado	208
Figura 8.25 - Camino paquetes lista negra	210
Figura 8.26 - Captura de tráfico ejemplo lista negra	211
Figura 8.27 - Reglas y contador de lista negra	211
Figura 8.28 - Resultado nmap ejemplo lista negra	212
Figura A.0.1 - Crear máquina virtual	217
Figura A.0.2 - Cargar imagen (iso) máquina virtual	218
Figura A.0.3 - Creación de usuario máquina virtual	218
Figura A.0.4 - Nombre de la máquina y ruta de los archivos	219
Figura A.0.5 - Espacio del disco máquina virtual	219
Figura A.0.6 - Personalizar hardware máquina virtual	220

Figura A.0.7 - Añadir <i>LAN segment</i> a la máquina virtual	220
Figura A.0.8 - Fin de la creación de la máquina virtual	221
Figura A.0.9 - Interfaz Wireshark	222
Figura A.0.10 - Interfaz TCPdump	222
Figura B-0.1 - Librerías nft(1)	227
Figura B.0.2 - Librerías nft(2)	227
Figura B.0.3 - nft instalado	228
Figura B.0.4 - Servicio nft	229
Figura C.0.1 - Módulos Conntrack-tools	232
Figura C.0.2 - Conntrack-tools instalado	232
Figura D.0.1 - Servicio firewalld	236
Figura D.0.2 - Estado Firewalld	236
Figura E.0.1 - Esquema escenario 2 equipos	243
Figura E.0.2 - Archivo ifcfg de PC1 (escenario 2 equipos)	245
Figura E.0.3 - Ip address PC1 (escenario 2 equipos)	245
Figura E.0.4 - Ip route PC1 (escenario 2 equipos)	246
Figura E.0.5 - Archivo interfaces Encaminador (escenario 2 equipos)	247
Figura E.0.6 - Ip address Encaminador (escenario 2 equipos)	248
Figura E.0.7 - Ip route (escenario 2 equipos)	248
Figura F.0.1 - Esquema escenario 3 equipos	249
Figura F.0.2 - Archivo ifcfg de PC1 (escenario 3 equipos)	251
Figura F.0.3 - Ip address PC1 (escenario 3 equipos)	252
Figura F.0.4 - Ip route PC1 (escenario 3 equipos)	252
Figura F.0.5 - Archivo ifcfg de PC2 (escenario 3 equipos)	254
Figura F.0.6 - Ip address PC2 (escenario 3 equipos)	254
Figura F.0.7 - Ip route PC2 (escenario 3 equipos)	254
Figura F.0.8 - Archivo interfaces Encaminador (escenario 3 equipos)	256
Figura F.0.9 - Ip address Encaminador (escenario 3 equipos)	257
Figura F.0.10 - Ip route Encaminador (escenario 3 equipos)	257
Figura F.0.11 - Esquema escenario NAT asimétrico	258
Figura F.0.12 - Ip address (Router NAT)	260
Figura F.0.13 - Ip route (Router NAT)	260
Figura H.0.1 - Establecimiento conexión TCP	273
Figura H.0.2 - Cierre conexión TCP	274

Nomenclatura

Hook	Enganche de netfilter para visualizar el tráfico
NIC	Tarjeta de Red del Equipo
Poc	Prove of Concepts (Prueba Conceptual)
Bridge	Puente Transparente
PC1	Equipo Usuario 1
PC2	Equipo Usuario 2
Encaminador	Equipo con Server Apache + nftables
Router NAT	Equipo usado para implementar el escenario de NAT Asimétrico
TCP	Transport Control Protocol
UDP	User Datagram Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol
SSH	Secure Shell
DNS	Domain Name System
FTP	File Transfer Protocol
IDS	Intrusion Detection System
IPS	Intrusion Prevention System

1 INTRODUCCIÓN

En la actualidad, casi todas las redes usan herramientas *software* o equipos *firewall* para proteger sus datos y controlar el acceso a los mismos.

Nftables[1] es el nuevo *framework* desarrollado por Netfilter como el sucesor natural de Iptables[2].

Pero poco sabemos acerca de los cambios que este ha introducido del estado del proyecto, de si está preparada para cubrir todas las funciones de iptables y que así podamos dejar esta atrás.

En este proyecto pretendo dar una visión ampliada acerca de la herramienta nftables, así como estudiar el funcionamiento y la configuración de esta.

1.1 Motivación del Proyecto

Este proyecto surge de la necesidad de documentar una herramienta que se presenta como la sucesora de iptables, herramienta que ha sido usada ampliamente en los sistemas Unix[3].

Nftables posee una potencia mayor a la hora de trabajar con los paquetes y de realizar el filtrado de los mismos, objetivo que ha conseguido al mejorar los módulos con los que trabaja, pero por desgracia la cantidad de documentación oficial y explicaciones del funcionamiento de esta son extremadamente escasos.

Por ello planteamos el proyecto como una guía de usuario aplicada, de forma que cualquier lector que tenga una base de conocimientos de Linux, pero que desconozca el funcionamiento de un *firewall* o de Netfilter, pueda, a través de estas páginas, encontrar un punto de partida sólido que le permita, no solo conocer la base de esta herramienta, si no, la forma en la que procesan los paquetes y los módulos que nos permiten trabajar con ellos.

También se puede comprobar, que a diferencia de iptables, esta herramienta carece de laboratorios de pruebas, que nos permitan replicar escenarios o probar las funcionalidades que nos ofrece.

Es por ello que se plantea que la guía pueda servir para que cualquiera pueda, siguiendo los pasos que en ella se indican, montar un pequeño laboratorio que le permita poner a prueba la herramienta, así como simular los escenarios de prueba que aquí se realizan y comprobar las salidas y resultados de los mismos.

1.2 Estructura del Proyecto

La estructura que sigue este documento es la siguiente;

1. Comenzamos realizando un estudio del estado del arte, en el cual abordaremos la versión actual de la herramienta, que tipo de herramientas existen en el mercado tanto de software libre como de software propietarios, el estado actual de sus actualizaciones y los puntos que cubren.
2. Después, se procederá a realizar una primera aproximación a la herramienta mediante la explicación de su uso básico, es decir, las librerías que la conforman, el uso de las mismas, los comandos básicos para la creación del armazón que usaremos para aplicar las reglas que creamos y detalles del funcionamiento, es decir, como se trabaja en el núcleo de Linux con los paquetes de red que recibimos por nuestra NIC, donde se guardan, etc...

3. En este punto se estudiará el camino natural que siguen los paquetes tanto en la herramienta como en su tránsito por el núcleo de Linux.
4. El siguiente punto del trabajo pretenderá darnos una visión de qué son los managers, como funcionan y el análisis del uso básico de uno de ellos, en este caso el elegido ha sido Firewalld, del cual hablaremos más adelante.
5. Los siguientes tres puntos, se basarán en explicar y dar una visión amplia del uso de nftables con sus tres principales módulos/usos, el seguimiento de conexiones (conntrack), el filtrado de paquetes que atravesen nuestro sistema firewall, y la traducción de direcciones (NAT).

Esto lo conseguiremos mediante el planteamiento de escenarios de pruebas a través de los cuales pretendemos ampliar la visión del uso de los mismos, así como plantear ejemplos y pruebas que el lector podrá simular para comprobar el correcto funcionamiento, dando así la base que permita probar nuevas funcionalidades que puedan surgir en un futuro.

La estructura de estos tres puntos será la siguiente:

- Se realizará una primera aproximación al módulo a analizar, comandos, funcionalidades, etc...
- Se planteará la creación de un escenario de pruebas el cuál se configurará paso a paso en los Anexos de este documento para crear la estructura general y en el capítulo correspondiente plantearemos los cambios más específicos.
- Se hará la realización de pruebas sobre el mismo, adjuntando capturas de pantalla de la configuración de la herramienta, de los logs generados, así como del sniffer de paquetes.

Como sniffer, se decide usar Wireshark, para realizar la captura de tráfico en el equipo Encaminador.

6. Y por último se incluirá un punto que contendrá las conclusiones sacadas de la realización de este proyecto, así como la opinión de la herramienta tras explorar a fondo el uso de la misma.

A lo largo del proyecto, se realizan referencias a diferentes webs o pdfs, consultados para la investigación de este proyecto. En este punto, se debe hacer una advertencia y es que la mayoría de esos documentos o webs tiene una cantidad considerable de inexactitudes con respecto al funcionamiento de Netfilter o de su *framework*.

Por ello se aconseja al lector que tome estas fuentes con cautela, ya que diferentes referencias pueden indicar comportamientos distintos de la misma herramienta.

1.3 Alcance y Objetivos

El alcance de este proyecto busca desarrollar una guía que sirva como base para el entendimiento y puesta en funcionamiento de la herramienta nftables.

Buscamos también la creación de escenarios de prueba que permitan comprender la actuación de las reglas, así como el análisis de los módulos más importantes de esta herramienta.

Al ser nftables un proyecto que se encuentra vigente y que va actualizándose con cierta asiduidad, se ha determinado tomar una versión concreta de este y trabajar sobre ella, por lo cual, se dejará clara la versión de la herramienta usada, así como las versiones de compilación del kernel y de los sistemas operativos usados.

Al ser un trabajo fin de grado, el análisis exhaustivo de la herramienta con su funcionalidad completa se escapa a los límites de este proyecto, por ello el objetivo, es que el lector tome esta guía como punto de partida para el entendimiento de Netfilter y de su framework Nftables, así como mostrarle ejemplos de los diferentes tipos de reglas y módulos que se pueden usar, basándonos en escenarios simples que no añadan complejidad a la comprensión del mismo.

2 ESTADO DEL ARTE

Nftables es un proyecto que pertenece a Netfilter, el cual busca desarrollar el framework que será el sustituto de iptables.

Ambos proyectos son herramientas que nos permiten tener en nuestro sistema Unix un *firewall* de paquetes de nivel 4 (capa de transporte).

Actualmente la última versión estable lanzada de nftables es la 0.9.7 lanzada el 27 de octubre de 2020, mientras que la de iptables es la versión 1.8.6 lanzada el 31 de octubre de 2020 tal como se muestra en la Figura 2.1.

2020-10-31

iptables 1.8.6 released

The Netfilter Core Team has released [iptables-1.8.6](#).

2020-10-27

nftables 0.9.7 released

The Netfilter Core Team has released [nftables-0.9.7](#).

Figura 2.1- Últimas versiones Herramientas Netfilter

2.1 Estado actual de la herramienta

Ahora mismo la página de Netfilter nos indica que iptables está siendo reemplazada, y recomiendan la migración a su nueva herramienta nftables, así mismo, esto también ocurre con herramientas como ebtables[4] y arptables[5].

A pesar de esto, estas herramientas todavía reciben actualizaciones y parches de soporte, aunque como se indica, al estar marcadas como descontinuadas, los parches cada vez son menos frecuentes y piden la migración a su nueva herramienta, la cual engloba las funcionalidades de las herramientas anteriormente mencionadas.

Aunque nftables es el sustituto natural y al que ya se recomienda su migración, existe una falta de documentación oficial de esta herramienta, situación que no ocurría con las herramientas anteriores como bien se puede comprobar.

Este problema está siendo subsanado y se busca cada vez más documentar de forma más completa la wiki de la herramienta, la cual contiene explicaciones muy breves de algunos módulos, así como una descripción de las novedades que la herramienta incorpora en sus nuevas versiones.

Este proyecto servirá del mismo modo para complementar y ampliar la información de la wiki.

La wiki[6] además contiene una pequeña guía para instalar la herramienta mediante su código fuente, así como algunos ejemplos “cortos” de comandos y reglas para nftables.

Nftables ya se encuentra disponible en las últimas versiones de los sistemas Unix.

En S.O RedHat[7], el framework de nftables viene instalado de forma nativa, debido a que la gran mayoría de usuarios sigue usando iptables, esta viene con la implementación de la capa de adaptación, la cual permite escribir nuestras reglas en iptables y que esta se encargue de traducirlas a sintaxis de nftables.

Pero, ¿a qué es debido que esté costando tanto la transición de iptables a nftables?

Tanto los usuarios como los desarrolladores de managers o herramientas de gestión de reglas, tienen su código preparado para trabajar con iptables, el hecho de que nftables abandone esa sintaxis para abrazar una más intuitiva, como es la usada por tcpdump, ha generado parte de rechazo.

Este cambio de sintaxis, aunque resulte más amistoso para el usuario, implica la actualización de muchas herramientas, impidiendo que estas puedan actualizarse mediante el uso de parches, ya que en la mayoría de casos implica la creación de una nueva herramienta, partiendo de la generación de un nuevo código que acepte esta nueva sintaxis.

Es por ello que managers ampliamente utilizados por la comunidad como Shorewall, Ipcop, Ipfir, han decidido no migrar.

Existen otros proyectos que ya han comenzado la migración, permitiendo trabajar con iptables y nftables, un caso de estos es Suricata[9] y Firewalld[10], aunque como hemos mencionado anteriormente, carecen de explicaciones o información que ilumine esta transición, encontrándose la gran mayoría de sus documentos basados en iptables.

Suricata apenas concede información acerca de la manipulación de reglas y creación de estas con nftables, de hecho, si accedemos a sus documentos y sus guías, en ellas aparece explicado con detalle el uso que se realiza con iptables y solo una primera aproximación con nftables.

Lo mismo ocurre en este caso con Firewalld, a pesar de indicar que la herramienta puede trabajar de forma nativa con nftables, si accedemos a la documentación de la misma podemos darnos cuenta que toda la guía viene realizada en base a iptables y que apenas se encuentran menciones de nftables.

Esta es una de las razones principales de la lenta migración que se está produciendo hacia esta nueva herramienta.

2.2 Estudio de Mercado

En esta sección vamos a realizar el estudio de mercado de los firewalls en la actualidad.

Debemos saber que, aunque nosotros usamos herramientas de software libre como puede ser iptables/ipsec/nftables etc..., no son las más usadas en ambientes profesionales o de negocios.

Las grandes empresas suelen decantarse por equipos hardware que realizan la función de *firewall* y por código propietario antes que por software libre.

Es por ello que, en la mayoría de los casos, es habitual encontrarnos con *firewalls* de empresas tales como Fortinet[12], Cisco[13], Zyxel[14], Palo Alto[15] etc...

Esto se debe a que estas empresas ofrecen un mejor soporte y seguimiento del producto de lo que podemos obtener con herramientas de software libre.

Es verdad que seguimos encontrándonos a estas herramientas en pequeños escenarios como pueden ser servidores de pequeñas empresas, laboratorios, clases etc..., pero lo normal, es que una empresa quiera cierta seguridad a la hora de proteger su entorno.

A pesar de que con el software libre podemos conseguirlo, necesitamos de una buena configuración, la cual, requiere una buena inversión de tiempo.

Los firewalls de empresas como las mencionadas anteriormente, ocultan la complejidad de la configuración y de la inversión de tiempo, que requieren los firewalls de software libre, es por ello que las empresas se decantan por esta solución, teniendo por contrapunto el costo de las mismas.

Estos productos no suelen ser baratos y van desde precios tales como los 500 euros del producto más básico de Fortinet (Fortigate 60F), un dispositivo que nos permite administrar la seguridad de una red con un total de 10 puertos, hasta unos 6.459 euros por un equipo *firewall* de Cisco (ASR1000-2T).



Figura 2.2 - (Fortigate 60F)

Por ello muchas empresas realizan auditorías para saber el nivel de seguridad necesario en sus redes y adaptar una solución acorde a las necesidades de esta.

Estos equipos no solo proveen del sistema hardware, sino que cada uno tienen su propio sistema software para la inspección de paquetes, toma de decisiones, seguimiento de conexiones etc...

En nuestro caso Netfilter provee una solución software (Nftables) que permite ser instalada en un equipo con una distribución Linux (Debian, Ubuntu, Kali, CentOS ...) y hacer que este equipo funcione como *firewall* en nuestra red.

Por lo cual, podemos observar, porqué este tipo de soluciones no son adoptadas por medianas/grandes empresas, que en su mayoría prefieren recurrir a soluciones más completas y que ya provean el equipo, dando un mayor soporte del que se puede conseguir con esta herramienta.

3 NFTABLES INTRODUCCIÓN

En el siguiente capítulo se analizarán las partes que conforman la herramienta nftables, así como conocimientos que se deben aclarar acerca de cómo se realiza la obtención de los diferentes paquetes de red y comandos básicos para crear las estructuras, en las que más tarde se profundizará.

Lo primero que se va a ver, es la instalación de la herramienta, en este apartado se explicará la instalación usando los repositorios de nuestro sistema Debian[16] y más adelante, en el “Anexo B - Instalación Nftables”, veremos la instalación desde código fuente para un sistema Debian.

Se recomienda revisar del mismo el “Anexo H - Repaso Protocolos (TCP,UDP,ICMP)”, en el cual, se realiza un repaso de los conceptos básicos de los protocolos y sus mensajes.

3.1 Instalación Nftables

La instalación de esta herramienta se llevará a cabo en el equipo Encaminador y a continuación se detallarán los pasos para realizarla, así como las comprobaciones de cada uno de los pasos para asegurarnos de que la instalación se realiza correctamente.

1. **[Equipo:Encaminador - Usuario:dit]** Para la instalación, lo primero que se debe realizar es la actualización del kernel, para ello se ejecutan los siguientes comandos:

```
Equipo:Encaminador Usuario:dit
```

```
$ sudo apt-get update  
$ sudo apt-get upgrade
```

Con esto ya se tendrá el sistema actualizado, para comprobarlo se puede hacer uso del siguiente comando, el cual imprimirá la información del sistema y la versión de su kernel, pudiendo observar en la *Figura 3.1* el resultado.

```
Equipo:Encaminador Usuario:dit
```

```
$ hostnamectl
```

```
dit@linux: ~  
Archivo Editar Ver Buscar Terminal Ayuda  
dit@linux:~$ hostnamectl  
    Static hostname: localhost  
    Transient hostname: linux  
        Icon name: computer-vm  
        Chassis: vm  
    Machine ID: ac0b4f46f3a4450bac7f5af4aa36b0d3  
    Boot ID: 0247793146fc4a8ea8990c87f132f5a2  
Virtualization: vmware  
Operating System: Ubuntu 18.04.5 LTS  
Kernel: Linux 4.15.0-129-generic  
Architecture: x86_64  
dit@linux:~$ uname -a  
Linux linux 4.15.0-129-generic #132-Ubuntu SMP Thu Dec 10 14:02:26 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux  
dit@linux:~$
```

Figura 3.1 - Versión del kernel

En la Figura 3.1, se puede comprobar que se usa una distribución de Linux - Debian 18.04.5 y que la versión del kernel es 4.15.0-129-generic[17].

Una vez llegados a este punto, se tendrá que tener en cuenta las dependencias de la herramienta, que no son más que los módulos o las bibliotecas necesarias para el correcto funcionamiento de la misma.

En este caso se debe tener instalados en el equipo los siguientes paquetes:

- **Libmnl [18]:** Es una librería usada por Netfilter para el desarrollo de su *framework*, la cual tiene como principal ventaja, evitar la complejidad de las abstracciones a la hora de trabajar con los elementos del núcleo de Netfilter.

Por lo que esta librería provee de los helpers necesarios para conseguir ese nivel de abstracción.

- **Libnftnl [19]:** Esta librería provee de una API de bajo nivel a Netlinks, siendo esta librería la que permite la comunicación entre la API y el módulo de Netlinks.
- **Libgmp [20]:** Es un paquete que nos permite operar con una librería matemática para la manipulación de enteros, flotantes, entre otros tipos de datos.
- **Libreadline [21]:** Esta librería nos proporciona las funciones necesarias para que nuestro programa pueda interpretar los comandos escritos por la entrada estándar (la línea de comandos).

Estas son las principales dependencias de la herramienta como se podrá ver en el “Anexo B - Instalación Nftables”, estas a su vez, dependen de otros módulos que deberemos descargar e instalar. Todo esto se encuentra indicado en su anexo.

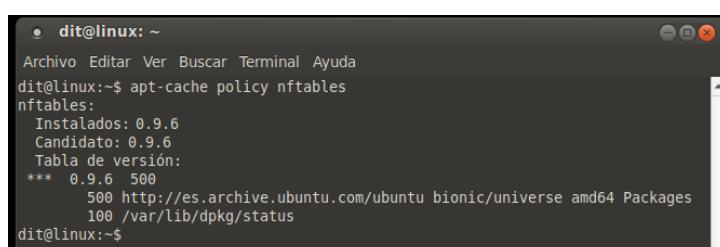
2. **[Equipo:Encaminador - Usuario:dit]** Se procede entonces a la instalación usando repositorios.

```
Equipo:Encaminador Usuario:dit
$ sudo apt-get install nftables
```

Una vez escrito el comando, se procederá a la instalación de nftables, se debe tener en cuenta que hay sistemas que ya traen de manera nativa esta herramienta instalada, en caso de que el sistema no la traiga, se procederá siguiendo estos sencillos pasos para instalar la herramienta en el sistema.

3. **[Equipo:Encaminador - Usuario:dit]** Para comprobar que la instalación se ha llevado a cabo correctamente, se hace uso del siguiente comando y se comprueba que la salida coincida con la mostrada (ver Figura 3.2).

```
Equipo:Encaminador Usuario:root
$ apt-cache policy nftables
```



The screenshot shows a terminal window with the following text:
Equipo:Encaminador Usuario:root
\$ apt-cache policy nftables
nftables:
 Instalados: 0.9.6
 Candidato: 0.9.6
 Tabla de versión:
*** 0.9.6 500
 500 http://es.archive.ubuntu.com/ubuntu bionic/universe amd64 Packages
 100 /var/lib/dpkg/status
dit@linux:~\$

Figura 3.2 - Comprobación instalación

4. [Equipo:Encaminador - Usuario:dit] Una vez comprobado que se ha instalado, se tendrá que comprobar que los módulos se han cargado correctamente, para ello, se hace uso del siguiente comando.

```
$ grep CONFIG_NFT_ /boot/config-[versión del kernel tengamos, en nuestro caso, 4.15.0-129-generic]
```

Equipo: Encaminador	Usuario: root
<pre>\$ grep CONFIG_NFT_ /boot/config-4.15.0-129-generic</pre>	

```
dit@linux:~$ grep CONFIG_NFT_ /boot/config-4.15.0-129-generic
CONFIG_NFT_EXTHDR=m
CONFIG_NFT_META=m
CONFIG_NFT_RT=m
CONFIG_NFT_NUMGEN=m
CONFIG_NFT_CT=m
CONFIG_NFT_SET_RBTREE=m
CONFIG_NFT_SET_HASH=m
CONFIG_NFT_SET_BITMAP=m
CONFIG_NFT_COUNTER=m
CONFIG_NFT_LOG=m
CONFIG_NFT_LIMIT=m
CONFIG_NFT_MASQ=m
CONFIG_NFT_REDIR=m
CONFIG_NFT_NAT=m
CONFIG_NFT_OBJREF=m
CONFIG_NFT_QUEUE=m
CONFIG_NFT_QUOTA=m
CONFIG_NFT_REJECT=m
CONFIG_NFT_REJECT_INET=m
CONFIG_NFT_COMPAT=m
CONFIG_NFT_HASH=m
CONFIG_NFT_FIB=m
CONFIG_NFT_FIB_INET=m
CONFIG_NFT_DUP_NETDEV=m
CONFIG_NFT_FWD_NETDEV=m
CONFIG_NFT_FIB_NETDEV=m
CONFIG_NFT_CHAIN_ROUTE_IPV4=m
CONFIG_NFT_REJECT_IPV4=m
CONFIG_NFT_DUP_IPV4=m
CONFIG_NFT_FIB_IPV4=m
CONFIG_NFT_CHAIN_NAT_IPV4=m
CONFIG_NFT_MASQ_IPV4=m
CONFIG_NFT_REDIR_IPV4=m
CONFIG_NFT_CHAIN_ROUTE_IPV6=m
CONFIG_NFT_CHAIN_NAT_IPV6=m
CONFIG_NFT_MASQ_IPV6=m
CONFIG_NFT_REDIR_IPV6=m
CONFIG_NFT_REJECT_IPV6=m
CONFIG_NFT_DUP_IPV6=m
CONFIG_NFT_FIB_IPV6=m
CONFIG_NFT_BRIDGE_META=m
CONFIG_NFT_BRIDGE_REJECT=m
dit@linux:~$
```

Figura 3.3 - Módulos cargados en el kernel

En la Figura 3.3 se puede ver que los módulos principales se han cargado correctamente y que no hay ninguno que haya dado error o que presente alguna incompatibilidad con el sistema.

Una vez realizada la instalación de la herramienta se procederá a comprobar su archivo de configuración.

El archivo de configuración de nftables se encuentra en la ruta: `/etc/nftables.conf`.

Este archivo permite guardar la estructura de tablas, cadenas y reglas que serán cargadas en el momento en el que se inicie el equipo.

Por otra parte, para poder ver los módulos que se encuentran cargados actualmente en la herramienta se ha de usar el siguiente comando.

Equipo: Encaminador	Usuario: dit
<pre>\$ lsmod grep nf_</pre>	

Con los módulos se pueden realizar dos operaciones, o cargarlos para ser usados o liberarlos.

- [Equipo:Encaminador - Usuario:root] Para cargar un módulo concreto se usa el siguiente comando:

```
Equipo:Encaminador Usuario:root
```

```
$ modprobe nombre_módulo
```

- [Equipo:Encaminador - Usuario:root] Para liberar un módulo concreto se usa el siguiente comando:

```
Equipo:Encaminador Usuario:root
```

```
$ rmmod nombre_módulo
```

Llegados a este punto, se tendrá listo nftables, por lo que el siguiente paso será aprender a configurar su base, permitiendo así tener una estructura a partir de la cual poder crear las reglas que usará el sistema para trabajar con los paquetes.

3.2 Construcción básica nftables

En este apartado se verá cómo preparar la estructura de tablas y cadenas para poder añadirle las reglas posteriormente.

Lo primero que puede preguntarse alguien que comience con el uso de nftables es, por qué es necesario preparar una estructura inicial para usarla, cuando en iptables, bastaba con instalarla para comenzar a trabajar con ella.

Y esto se debe a una explicación muy sencilla y es que en nftables no se tienen ni cadenas ni tablas definidas por defecto, de forma que depende totalmente del usuario el crearlas y configurarlas a su gusto para ser usadas.

Esto representa un avance, pero conlleva un inconveniente y es que se debe conocer cómo se establecen las cadenas y tablas para poder usarlas convenientemente, en caso contrario, es posible que las reglas creadas no vean nunca tráfico o que estas tengan un comportamiento inesperado.

Lo primero que se debe entender es el concepto de familias[22].

Las familias son las grandes agrupaciones en las cuales nftables divide el tráfico, es decir, dependiendo de la familia a la que se encuentre vinculada una tabla o cadena se podrá ver un tipo de tráfico u otro.

Por ello se detalla las familias existentes en la Tabla 1.

Familia	Descripción
<i>ip</i>	Esta familia es la principal y ve todo el tráfico Ipv4 que le llegue.
<i>ip6</i>	Esta familia permite centrarse solamente en aquel tráfico cuyo protocolo sea Ipv6.
<i>inet</i>	Esta familia se usa si se desea ver tanto tráfico Ipv4 como Ipv6.
<i>arp</i>	Esta familia, como su propio nombre indica, solo mostrará tráfico de tipo ARP.

bridge	Esta familia solo muestra aquellos paquetes que atravesarían un switch, es decir, solamente llega a nivel 2.
netdev	<p>Esta familia es especial, ya que permite trabajar con el tráfico que el driver de nuestra NIC acabe de pasar a la pila de red, de forma que esta familia será la primera en ver los paquetes del equipo.</p> <p>Esta familia es exclusiva de nftables y no tiene traducción en iptables.</p>

Tabla 1 - Familias nftables

Como se acaba de comprobar, las familias no son más que los diferentes tipos de tráfico que se puede encontrar en nuestra red y la explicación del uso de las mismas se debe a que con iptables, solo se podía ver tráfico Ip.

Si se deseaba ver otro tipo de tráfico, se debía usar otras herramientas tales como ebtables, la cual permitía ver el tráfico bridge, o arptables, la cual mostraba el tráfico arp que llegaba hasta el equipo.

Las familias nacen de la necesidad de agrupar todas las herramientas usadas con iptables para tratar diferentes tipos de tráfico en una sola herramienta.

Una vez se ha entendido este punto se pasará a ver la sintaxis básica de la herramienta.

Como ya se dijo anteriormente, la herramienta ha cambiado de sintaxis, por lo que los usuarios deben acostumbrarse a esta nueva manera de proceder.

Siendo la sintaxis general de la herramienta de la siguiente forma:

```
$ nft (Acciones) < Objeto > [Familia] < Nombre > [Opciones]
```

- **Acciones:** Representa la acción que se desea realizar, ya sea crear, añadir, listar, eliminar etc...
- **Objeto:** Representa el objeto o la parte sobre la cual se va a realizar la acción; una tabla, una cadena, una regla, un set...
- **Familia:** Aquí es donde se decide el tipo de familia a la que pertenece el objeto y como se vio anteriormente, esta es la parte en la que define el tipo de tráfico que tratará en el sistema.
- **Nombre:** El nombre que se le desea dar al objeto, como ya se vio con anterioridad, nftables no tiene ni cadenas ni tablas por defecto, así que pueden crearse y llamarse como el usuario desee.
- **Opciones:** Aquí se describirán las opciones que se pueden realizar con el objeto. Estas dependerán de a qué objeto se le esté aplicando, pues no tiene las mismas opciones una tabla, una cadena, una regla etc...

Se puede decir que esta es la estructura básica de un comando de nftables.

A continuación, se verá la creación de una tabla, que es la unidad principal para crear las estructuras de reglas en nftables.

3.2.1 Tablas

El comando que se usará para poder crearlas será el siguiente.

```
Equipo: Encaminador Usuario: dit
```

```
$ sudo nft create table ip Filter
```

Con este comando se crea una tabla, llamada Filter, la cual solo verá tráfico de tipo Ipv4 (esto queda determinado por la familia que se le indica a la tabla).

Cuando se crea una tabla, se puede usar, o bien la acción *add* o *create*, la diferencia es que *add* creará lo que se le diga, sin importar que esté ya creado, y *create* comprobará primero si este ya ha sido creado y en caso de que este ya exista simplemente no lo creará.

Es por ello que se recomienda el uso de la acción *create*, en lugar de la acción *add*.

En caso de que se quiera ver la tabla que se ha creado, se puede usar el siguiente comando, el cual imprimirá por pantalla todas las tablas que se tengan creadas, la salida del comando se puede observar en la *Figura 3.4*.

```
Equipo: Encaminador Usuario: dit
```

```
$ sudo nft list tables
```

```
root@linux:/home/dit# nft create table ip filter
root@linux:/home/dit# nft list tables
table ip filter
root@linux:/home/dit#
```

Figura 3.4 - Listar tablas

Una vez se ha creado una tabla, el siguiente paso es crear las cadenas que va a contener.

Para ello, primero se explicará los tipos de cadenas que se pueden tener y los *hooks* (enganches) a las que pueden asociarse.

3.2.2 Cadenas y Hooks

En este apartado se verá que tipos de cadenas existen y sus respectivos *hooks*.

Recordamos que a diferencia de iptables, en nftables no se tienen cadenas por defecto, sino que se tiene la libertad para crear las que se deseen.

Se debe tener en cuenta que existen dos tipos de cadenas:

- **Cadenas base:** Estas cadenas son las básicas que se pueden crear dentro del *framework* de nftables, su principal característica es que este tipo de cadenas “ve tráfico”, es decir, se encuentran asociadas a un determinado enganche (*hook*), el cual les permitirá en un cierto punto del transcurso del paquete por el sistema, poder verlo y tratarlo.

- **Cadenas de usuario:** Las cadenas de usuario a diferencia de las cadenas básicas no ven tráfico, esto se debe a que no están asociadas a ningún enganche. En este punto el usuario, puede preguntarse cuál es la utilidad de una cadena que no ve tráfico y es que, al igual que en iptables, estas cadenas son usadas para organizar el tráfico, o para dividirlo según sus características.

Una vez presentados los distintos tipos de cadenas que se tienen, se va proceder a explicar las cadenas base.

La sintaxis de este tipo de cadenas es la siguiente.

```
$ nft <acción> chain [family] <Nombre_tabla> <Nombre_cadena> { tipo
<tipo> hook <hook> priority <priority> ; [policy <policy> ;] }
```

En este punto se va a hacer un pequeño alto para explicar cómo ha de entenderse la sintaxis de las reglas a partir de ahora.

- []: Lo que se coloca entre corchetes, representa un elemento optativo, puede aparecer o no. En la mayoría de los casos debemos saber que si este no aparece, el elemento tomará un valor por defecto, en el caso anterior, la política (*policy*) puede o no aparecer, y en caso de que esta no lo hiciese su valor por defecto sería *accept*.
- { }: Lo que va entre este tipo de corchetes es obligatorio y representa las opciones de un comando, en el caso anterior se puede ver que para trabajar con la cadena se debe especificar su tipo, su *hook* y por último su prioridad.
- <>: Por último, lo que se coloca entre la diple quiere decir que es necesario que aparezca pero que el valor de este, lo determina el usuario, bien por que pueda tener varias opciones o porque es el usuario el que determina el nombre.

En el caso anterior por ejemplo se puede ver que se tiene <acción>, la cual es obligatorio que aparezca, pero es el usuario el que decide si el valor de esta es ; *add, create, flush, list...*, otro ejemplo es el caso de <Nombre_tabla>, el cual también debe aparecer y es el usuario el que determina el nombre que va a tener , por ejemplo, *Filter*.

3.2.2.1 Tipo de Cadenas Base

Comprendido como se debe entender la sintaxis de los comandos de nftables, se prosigue viendo el tipo de las cadenas base.

Como se puede ver en la sintaxis, para crear una cadena base, se debe especificar en el apartado de opciones, su tipo, el *hook* al cuál se suscribirá y la prioridad de la misma.

El tipo de cadena indica las posibles reglas que esta puede contener, teniendo 3 tipos distintos, los cuales pueden verse en la Tabla 2.

Tipo	Descripción
<i>Filter</i>	Este tipo de cadena nos sirve para filtrar el tráfico que llegue al <i>hook</i> al cuál estemos suscripto, filter, soporta tráfico de todas las familias que tenemos. (ip, ip6, inet, arp, bridge)
<i>Route</i>	Este tipo de cadenas podemos compararla con mangle en iptables y su misión es que nos permite volver a realizar la decisión de routing de un paquete. La diferencia con iptables es que este tipo de cadenas solo pueden estar suscritas al <i>hook output</i> , y solo puede ver tráfico de las familias (ip, ip6 e inet)

Nat	Este tipo de cadenas es la que utilizamos cuando queremos realizar traducción de direcciones mediante el uso de las opciones de nat. Este tipo de cadenas ve tráfico de las familias (ip, ip6 e inet)
------------	--

Tabla 2 - Tipos de cadenas

Con esto, queda visto los tipos de cadenas base que se pueden encontrar en la herramienta.

Lo siguiente que se verá, son los tipos de enganches (*hook's*) a los cuales pueden subscribirse las cadenas base para ver tráfico.

3.2.2.2 Tipos de Enganches (*Hooks*)

Un *hook* [23], es el mecanismo que utiliza Netfilter para asociar las cadenas base de su *framework*, a un tráfico determinado, en un punto concreto del camino que los paquetes siguen a lo largo del sistema.

Este camino será explicado en el capítulo “Flujo de Paquetes (Packet Flow)” de esta guía, por ahora bastará con comprender que son y cómo funcionan y más adelante se verá en qué puntos del código operan y como lo hacen.

Los tipos de *hooks* que se tienen, vienen explicados en la Tabla 3.

Enganche(<i>hook</i>)	Descripción
Prerouting	Este enganche es el primero que ve tráfico sin contar con <i>ingress</i> . En este punto el paquete acaba de llegar para procesarse y todavía no ha sido enrutado. Es el mejor punto para realizar el primer filtrado de paquetes según si estos van para nuestro equipo o si han de ser re-enrutados(<i>forwarding</i>)
Input	En el camino hacia un proceso local, este es el enganche que va después de <i>prerouting</i> . En este punto ya se ha tomado la decisión de <i>routing</i> y se ha visto que el destino del paquete es nuestra máquina, por lo que se dirige hacia un proceso local en esta.
Forward	Este enganche es el camino natural que sigue un paquete que ha llegado a nuestra máquina, pero que no se dirige a ella, de forma que necesita ser encaminado. Por ello, este enganche viene después de que se haya pasado por <i>prerouting</i> y se haya tomado la decisión de encaminamiento, determinando que el paquete no es para un proceso local y que por lo tanto ha de encaminarse hacia su destino. Este es un buen punto para filtrar aquellos paquetes que no deseamos que sean encaminados.
Output	Este enganche es el primero que ve un paquete que ha sido generado por un proceso local y el cual ya ha tomado una decisión de encaminamiento.
Postrouting	Es el último enganche que ve un paquete antes de salir por nuestra tarjeta NIC y abandonar nuestro equipo. Es el último punto en el que podemos trabajar con el paquete, y es el <i>hook</i> usado para realizar la traducción de dirección origen a nuestro paquete.

Ingress	Este enganche es nuevo y exclusivo de nftables. Este es el primer enganche que ve un paquete que ha llegado a nuestro equipo y que nos permite darle una tramitación muchísimo más rápida a un flujo o paquetes específicos.
----------------	---

Tabla 3 - Enganches Netfilter

Ya se ha visto las opciones que tienen las cadenas, el tipo que estas pueden tomar, el enganche al que pueden asociarse y por último, queda por ver y comprender como funciona la prioridad de estas.

3.2.2.3 Prioridades de las Cadenas

En iptables se ha visto que a la hora de procesar las reglas se seguía un orden, procesando primero las reglas de *mangle*, después *nat*, *filter*... etc.

En nftables es distinto, al crear el usuario las tablas y cadenas, si se tienen dos cadenas de distinto tipo que se encuentran asociadas al mismo enganche, ¿cuál se procesaría primero y cuál se haría después?

La respuesta oficial es que nftables procesaría primero las reglas de la última cadena que se haya creado y seguiría así por orden de antigüedad hasta llegar a la cadena más antigua.

Pero esto no es manera de proceder, ya que el usuario no tiene por qué acordarse del orden en el que se han ido creando las cadenas, o podría querer un orden distinto a este, es por ello que para ordenar su procesado se usa la prioridad.

La prioridad no es más que un valor de tipo *int* que indica el orden en el que se procesan las cadenas.

Los valores de la misma van desde el (-400) que sería la cadena con más prioridad del sistema, hasta el (300), cadena con menos prioridad.

En general existen unas prioridades predefinidas, que dan una idea acerca de a qué valor correspondería según la acción que se desea realizar, es por ello que en la Tabla 4, se recogen las prioridades más representativas para ser usadas por el usuario.

Nombre	Prioridad	Familia	Enganche
NF_IP_PRI_RAW	(-300)	ip, ip6, inet	Todos
NF_IP_PRI_CONNTRACK	(-200)	ip, ip6, inet	Todos
NF_IP_PRI_MANGLE	(-150)	ip, ip6, inet	Todos
NF_IP_PRI_NAT_DST	(-100)	ip, ip6, inet	Prerouting
NF_IP_PRI_FILTER	(0)	ip, ip6, inet, arp, bridge	Todos
NF_IP_PRI_NAT_SRC	(100)	ip, ip6, inet	Postrouting

Tabla 4 - Prioridades de las cadenas

Se puede comprobar que la cadena que más prioridad ha de tener es aquella que trabaja con el paquete en modo *raw*, es decir en crudo, sin diferenciación y sin utilizar las estructuras que recogen los datos del mismo.

Tras la prioridad de *raw*, interesa tener identificado el flujo de comunicación del paquete, por lo tanto, le sigue la prioridad de *Conntrack* (módulo de seguimiento de conexiones de Netfilter).

Después vendría *mangle*, el cual, permite la modificación de ciertos campos del paquete para su posterior filtrado/ inspección.

Y justamente antes de poder realizar el filtrado se puede observar la prioridad de *NAT-DST*, el NAT de destino tiene un valor de prioridad menor que el de filtrado, consiguiendo así que esta operación se realice antes del filtrado del paquete.

El tipo neutral de prioridad lo tienen las cadenas de filtrado, cuya cadena base cuenta con la prioridad 0.

Y, por último, el *NAT-SRC*, o NAT de origen.

Nos interesa que el valor de prioridad de este tipo de NAT sea el mayor posible, ya que esta ha de ser una de las últimas operaciones que se realice sobre el paquete, antes de que abandone el equipo por el *hook postrouting*.

Con esto, queda explicado por completo las opciones y configuración que se pueden hacer con las cadenas base, a continuación, se mostrará una explicación de cómo crear una cadena de usuario.

3.2.2.4 Cadena de usuario

Una cadena de usuario como se ha indicado al principio de este punto es aquella que no se encuentra asociada a ningún enganche, de forma que ella por sí sola no ve ningún tipo de tráfico.

Su principal uso es organizar y discernir tráfico y para ello usamos unas acciones de reglas que veremos en profundidad en el punto “Reglas básicas”, las cuales son; *jump* y *go to*.

Estas acciones permiten saltar desde una regla de una cadena a otra regla de otra cadena para que el paquete siga el procesado en esta, la diferencia básica de las dos acciones anteriormente mencionadas, es que con *jump*, después de procesar todas las reglas de la nueva cadena, se vuelve a la cadena base desde la que se saltó, mientras que con *go to*, terminamos el procesado en la cadena a la que vamos.

Para explicar el funcionamiento de las mismas veremos un ejemplo.

Ejemplo 1 cadena de usuario.

Descripción del Ejemplo:

En este ejemplo, se va a crear una cadena base llamada *output*, la cual estará asociada al enganche *output* y verá tráfico de tipo *inet* (Ipv4 e Ipv6).

De forma que en esta cadena, se creará una regla para que acepte el tráfico ICMP saliente del equipo Encaminador, para ello se realizará lo siguiente;

- La política por defecto será *accept*
- Si el tráfico de salida es ICMP, este saltará a la cadena de usuario que se crea, en la cual se contabilizará el número de paquetes ICMP que han salido del equipo Encaminador.

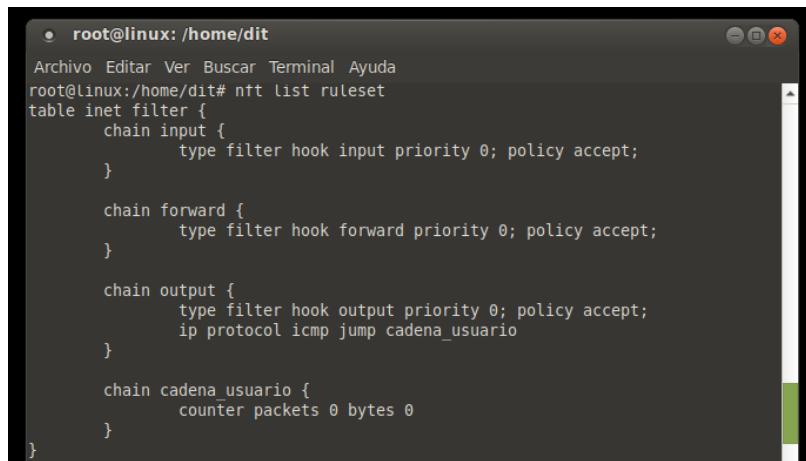
Análisis del ejemplo:

1. [Equipo:Encaminador - Usuario:root] Se crea la siguiente estructura de reglas en nftables.

```
Equipo:Encaminador Usuario:root

$ nft add table inet filter
$ nft add 'chain inet filter output { type filter hook output
priority 0; policy accept ; }'
$ nft add chain inet filter cadena_usuario
$ nft add rule inet filter output ip protocol icmp jump cadena
usuario
$ nft add rule inet filter cadena_usuario counter
```

Obteniendo la configuración vista en la Figura 3.5.



A terminal window titled 'root@linux: /home/dit' showing the output of the 'nft list ruleset' command. The output displays a table 'inet filter' with four chains: 'input', 'forward', 'output', and 'cadena_usuario'. The 'input' chain has a rule with priority 0, policy accept, and a type filter hook. The 'forward' and 'output' chains also have similar rules with priority 0, policy accept, and type filter hooks. The 'cadena_usuario' chain contains a single rule with a counter packet 0 bytes 0.

```
root@linux: /home/dit
Archivo Editar Ver Buscar Terminal Ayuda
root@linux:/home/dit# nft list ruleset
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
        ip protocol icmp jump cadena_usuario
    }

    chain cadena_usuario {
        counter packets 0 bytes 0
    }
}
```

Figura 3.5 - Ejemplo 1 cadena usuario

2. [Equipo:Encaminador - Usuario:dit] Se procede a realizar un ping a la dirección de la máquina pública 8.8.8.8 y se comprobará si este es capaz de completarse.

En otro terminal se imprimirá el *ruleset* de nftables y se comprobará si el contador ha aumentado en consecuencia al número de paquetes ICMP generados.

- Terminal 1:

```
Equipo:Encaminador Usuario:root

$ nft list ruleset
```

- Terminal 2:

```
Equipo:Encaminador Usuario:dit

$ ping -c 3 8.8.8.8
```

The image shows two terminal windows side-by-side. The left window, titled 'root@linux:/home/dit', displays the output of the command 'nft list ruleset'. It defines a table 'inet filter' with several chains: 'input', 'forward', 'output', and 'cadena_usuario'. The 'input' and 'forward' chains have a 'type filter hook input priority 0; policy accept;' rule. The 'output' chain has a similar rule. The 'cadena_usuario' chain contains a 'counter packets 3 bytes 252' rule. The right window, titled 'dit@linux:', shows the output of a 'ping' command to '8.8.8.8'. It shows three ICMP echo requests being sent with TTL=128 and round-trip times around 226 ms. The final line shows the ping statistics: 3 packets transmitted, 3 received, 0% packet loss, time 2005ms, rtt min/avg/max/mdev = 13.000/84.296/226.701/100.695 ms.

```

root@linux:/home/dit# nft list ruleset
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }
    chain forward {
        type filter hook forward priority 0; policy accept;
    }
    chain output {
        type filter hook output priority 0; policy accept;
        ip protocol icmp jump cadena_usuario
    }
    chain cadena_usuario {
        counter packets 3 bytes 252
    }
}
root@linux:/home/dit#

```

```

dit@linux:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=128 time=226 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=128 time=13.1 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=128 time=13.0 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 13.000/84.296/226.701/100.695 ms
dit@linux:~$ 

```

Figura 3.6 - Resultado ejemplo 1 cadena usuario

Como se puede ver en la Figura 3.6, en el terminal 2 se han lanzado tres paquetes ICMP y el contador de la regla del terminal 1 muestra que ha aumentado de forma correspondiente a esta acción.

Al ser la política por defecto *accept* se puede comprobar cómo el ping se completa.

El camino que sigue el paquete ICMP es el siguiente :

- 1) El paquete ICMP es generado por el kernel, se toma la decisión de *routing* del mismo y llega a la cadena output de la tabla filter.
- 2) El paquete hace *match* con la regla de output ya que el protocolo de este es ICMP, por lo que su acción es *jump* a la cadena_usuario
- 3) En la cadena_usuario aumenta en 1 el contador que se ha creado y tras aumentarlo vuelve a la cadena desde la que saltó (output)
- 4) En la cadena output al no quedar más reglas, se toma como acción la que indica la política por defecto (*accept*).
- 5) El paquete continúa por postrouting y sale del equipo Encaminador.

Ejemplo 2 cadena de usuario.

Descripción del Ejemplo:

En el siguiente ejemplo, se verá como discernir tráfico usando una cadena de usuario y de esta manera comprobar la utilidad principal de este tipo de cadenas.

En el equipo Encaminador se tiene tráfico TCP, UDP e ICMP y se busca que el tráfico ICMP de salida que tenga como destino la *Ip*: 8.8.8.8/32 sea denegado, mientras que el tráfico ICMP que tenga como destino la *Ip*: 1.1.1.1/32 sea aceptado, pero además se busca saber el total de tráfico ICMP generado (sea o no aceptado) y la cantidad por separado de tráfico aceptado y denegado.

Para evitar masificar con reglas la cadena base output, lo que se hace es utilizar la misma regla que en el ejemplo anterior y si el tráfico que atraviesa esta regla es de tipo ICMP, que salte a la cadena_usuario.

Análisis del ejemplo:

1. [Equipo:Encaminador - Usuario:root] En la cadena usuario se añadirán las reglas en el siguiente orden;
 - 1) Se añade la regla de denegación de tráfico ICMP con destino a la *Ip*: 8.8.8.8/32 y la contabilización de esta
 - 2) Se añade la regla de aceptación de tráfico ICMP con destino a la *Ip*: 1.1.1.1/32 y la contabilización de esta.

Para ello se usa de base, el esquema del ejemplo anterior y se añaden las reglas tal y como se ha indicado, quedando el *ruleset* como aparece en la Figura 3.7:

Equipo:Encaminador Usuario:root

```
$ nft add rule inet filter cadena_usuario ip daddr 8.8.8.8/32
count drop
$ nft add rule inet filter cadena_usuario ip daddr 1.1.1.1/32
count accept
```

Figura 3.7 - Ejemplo 2 cadena usuario

2. [Equipo:Encaminador - Usuario:dit] Una vez creadas las reglas, se realizan los siguientes pasos.

- Desde el terminal 1 se comprueba el *ruleset* de nftables.

Equipo:Encaminador Usuario:root

```
$ nft list ruleset
```

- Desde el terminal 2 se realizan dos envíos ICMP con los siguientes comandos.

Equipo:Encaminador Usuario:dit

```
$ ping -c 3 1.1.1.1
$ ping -c 3 8.8.8.8
```

```

root@linux:/home/dit# nft list ruleset
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
        ip protocol icmp jump cadena_usuario
    }

    chain cadena_usuario {
        counter packets 11 bytes 924
        ip daddr 8.8.8.8 counter packets 3 bytes 252 drop
        ip daddr 1.1.1.1 counter packets 3 bytes 252 accept
    }
}
root@linux:/home/dit# 

dit@linux: ~
Archivo Editar Ver Buscar Terminal Ayuda
^C
--- 1.1.1.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1016ms
rtt min/avg/max/mdev = 5.473/114.632/223.791/109.159 ms
dit@linux:~$ ping 1.1.1.1
PING 1.1.1.1 (1.1.1.1) 56(84) bytes of data.
64 bytes from 1.1.1.1: icmp_seq=1 ttl=128 time=7.32 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=128 time=29.1 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=128 time=8.86 ms
^C
--- 1.1.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 7.324/15.122/29.178/9.958 ms
dit@linux:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
ping: sendmsg: Operación no permitida
ping: sendmsg: Operación no permitida
ping: sendmsg: Operación no permitida
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2045ms
dit@linux:~$ 

```

Figura 3.8 - Resultado Ejemplo 2 cadena usuario

Como se puede ver en la Figura 3.8, se ha aceptado solo el tráfico de la Ip: *1.1.1.1/32* mientras que el de la Ip: *8.8.8.8/32* se ha denegado.

Además, se puede observar cómo se ha contabilizado todo el tráfico ICMP y los datos de ambas Ip's por separado.

El camino que sigue el paquete ICMP con destino 1.1.1.1 es el siguiente:

- 1) El paquete ICMP es generado por el kernel, se toma la decisión de routing(destino 1.1.1.1) del mismo y este llega a la cadena output de la tabla filter.
- 2) El paquete hace *match* con la regla de output ya que el protocolo de este es ICMP, por lo que su acción es *jump* a la cadena_usuario
- 3) En la cadena_usuario aumenta en 1 el contador de la primera regla y hace *match* con la tercera regla, aumentando el contador de esta y tomando la acción de *accept* sobre el paquete.
- 4) El paquete continúa por postrouting y sale del equipo Encaminador.

El camino que sigue el paquete ICMP con destino 8.8.8.8 es el siguiente:

- 1) El paquete ICMP es generado por el kernel, se toma la decisión de routing(destino 8.8.8.8) del mismo y este llega a la cadena output de la tabla filter.
- 2) El paquete hace *match* con la regla de output ya que el protocolo de este es ICMP, por lo que su acción es *jump* a la cadena_usuario
- 3) En la cadena_usuario aumenta en 1 el contador de la primera regla y hace *match* con la segunda regla, aumentando el contador de esta y tomando la acción de *drop* sobre el paquete.

Conclusión del ejemplo:

Como se ha podido ver en los dos ejemplos anteriores, las cadenas de usuarios son usadas, principalmente, para realizar el tratamiento de un tráfico específico en una cadena.

La mayoría de ejemplos que se pueden encontrar de su uso, es para discernir tráfico por protocolos, por servicios(puerto destino) o en función de las direcciones de las cuales provenga el paquete, bien si este viene de la red interna, de Internet, de la DMZ etc...

3.2.3 Reglas básicas

Ya se ha podido comprobar el funcionamiento de las tablas, las cadenas y sus enganches, por último, queda por comprender como funciona la creación de reglas dentro de nftables.

En primer lugar, se verá cuál es la sintaxis de las reglas de nftables ya que, como se indicó en el comienzo de esta guía, esta se ha visto modificada con respecto a la usada en iptables.

```
$ nft <acción> rule [<famiy>] <Nombre_tabla> <Nombre_cadena> [handle <value>] [<selectores>] [<acciones>] [comment "Comentario"]
```

Esta es la sintaxis por defecto para las reglas.

Debemos indicar que la tabla, la cadena y la familia a la que pertenece la regla, deberá coincidir con la de la cadena a la que se la añade.

Si se desea en vez de añadir una regla, insertarla en una posición determinada, se usará la opción de *handler*.

El *handler* [24] o manejador, no es más que un número que indica la posición de la regla dentro de la estructura de nftables, de esta forma, si se desea insertar una regla antes que otra ya creada, basta con indicar el *handler* que posee esa regla ya creada, para saber en qué punto se desea añadir la próxima regla.

Los selectores son las opciones que proporcionan los módulos para especificar el tipo de tráfico que se desea tratar.

La parte de las acciones, es donde se indica como se debe proceder con el tráfico que haya coincidido con nuestra regla, se verá que existen dos tipos de acciones, las terminales y las no terminales, las cuales se explicarán en este mismo punto.

Y, por último, se da la opción de añadir un comentario para que quede constancia en el archivo, esto no es más que una explicación o indicación acerca de la regla.

3.2.3.1 Adaptación de reglas

En el punto anterior, se ha visto la sintaxis que tiene nftables para crear una regla, pero esta herramienta también ofrece la opción de usar la capa de adaptación para traducir las reglas a sintaxis de nftables.

Esta capa de adaptación no es más que una ayuda que permite traducir las reglas que usen la sintaxis de iptables, a sintaxis de nftables.

Las opciones de traducción son:

- Iptables-nft
- Ip6tables-nft
- Ebttables-nft
- Arptables-nft

Estos comandos que permiten traducir las reglas de iptables a nftables, se encargan de crear las tablas y cadenas necesarias para albergarlas, ocultando la complejidad de creación de estas al usuario.

En la Figura 3.9, se puede observar un ejemplo de traducción de regla de iptables a nftables, usando la opción *translate*.

```

root@linux:/home/dit# iptables-translate -A INPUT -p tcp --dport 80 -j ACCEPT
nft add rule ip filter INPUT tcp dport 80 counter accept
root@linux:/home/dit#

```

Figura 3.9 - Iptables translate

En el punto de “Filtrado de paquetes”, se analizarán los módulos de nftables que permiten la inspección de los paquetes y los valores de estos.

En este punto se analizarán las acciones que pueden realizar las reglas y los llamados objetos de estado.

El seguimiento de conexiones también se verá en el punto de “Seguimiento de Conexiones(Conntrack)”.

3.2.3.2 Acciones terminales

Como se ha comentado antes, las reglas tienen dos tipos de acciones, las que se consideran terminales y las no terminales.

Las terminales son aquellas que ponen fin a la evaluación del paquete con respecto del resto de reglas, tomando una decisión de como se ha de proceder con el paquete/tráfico que ha coincidido con ella.

Estas acciones quedan recogidas en la Tabla 5.

Acciones terminales	Descripción
<i>Accept</i>	Permite que el paquete sea aceptado.
<i>Drop</i>	Desecha el paquete y acaba la evaluación.
<i>Reject</i>	Rechaza el paquete y envía al origen del mismo un paquete de tipo ICMP informando de la acción. Si no se puede realizar la acción de <i>reject</i> por algún motivo, esta se toma como Drop.
<i>Queue</i>	Envía el paquete a una cola en el espacio de trabajo del usuario, acabando así la evaluación del resto de reglas.
<i>Continue</i>	Prosigue la evaluación con la siguiente regla de la cadena.
<i>Return</i>	Deja de evaluar las reglas de la cadena donde se encuentra y vuelve a la cadena desde la que se realiza el salto(jump). Si esta acción la colocamos en una cadena base, es igual que la acción Accept.
<i>Jump <chain></i>	Salta a la cadena de usuario que se le especifique y al terminar la evaluación de reglas en la misma vuelve a la cadena base desde la que se realizó el salto.
<i>Goto <chain></i>	Esta acción sirve para saltar a la cadena especificada, en la cual se terminará la evaluación del paquete sin volver a la cadena desde la que se realizó el salto.

Tabla 5 - Acciones terminales reglas

3.2.3.2.1 Acción Reject

A continuación, se podrá ver un ejemplo de estas acciones terminales.

En el punto anterior, se ha visto el uso de *jump* y de cómo poder sacarle partido con las cadenas de usuarios, así como el uso de *drop* y *accept* dentro de las reglas de las mismas.

En el siguiente ejemplo se mostrará el uso del *reject* de paquete, y el tipo de mensajes ICMP que este puede enviar.

Ejemplo reject.

Descripción del Ejemplo:

En este ejemplo se realizará una prueba con el equipo Encaminador, en el cual se pretende que haga *reject* a todo el tráfico entrante cuyo puerto origen sea el 80.

Se parte del ejemplo usado en “Cadena de usuario” y de la estructura creada en este.

Análisis del ejemplo:

1. [Equipo:Encaminador - Usuario:root] Se añade la siguiente regla a nftables y se activa Wireshark para escuchar los paquetes en la interfaz “ens33”.

Equipo:Encaminador Usuario:root

```
$ nft add rule inet filter input tcp sport 80 reject with icmp type host-unreachable
```

Con esta regla, se indica que cuando en la cadena *input* se reciba un paquete TCP, cuyo puerto de origen sea el 80, se descartará y se enviará un mensaje ICMP de tipo *host unreachable* de vuelta al origen.

En la Figura 3.10, se puede ver la estructura de reglas usada.

```
root@linux:/home/dit# nft list ruleset -a
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
        tcp sport http reject with icmp type host-unreachable # handle 16
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
        ip protocol icmp jump cadena_usuario # handle 6
    }

    chain cadena_usuario {
        counter packets 623 bytes 45255 # handle 8
        ip daddr 8.8.8.8 counter packets 3 bytes 252 drop # handle 9
        ip daddr 1.1.1.1 counter packets 9 bytes 756 accept # handle 10
    }
}
```

Figura 3.10 - Ejemplo reject

2. [Equipo:Encaminador - Usuario:dit] Desde el equipo Encaminador, se realiza una petición a la web de trajano, cuya Ip es: 193.147.162.130/32.

Equipo:Encaminador **Usuario:**root

```
$ curl 193.147.162.130
```

3. [Equipo:Encaminador - Usuario:root] Se accede a Wireshark para observar los paquetes generados.

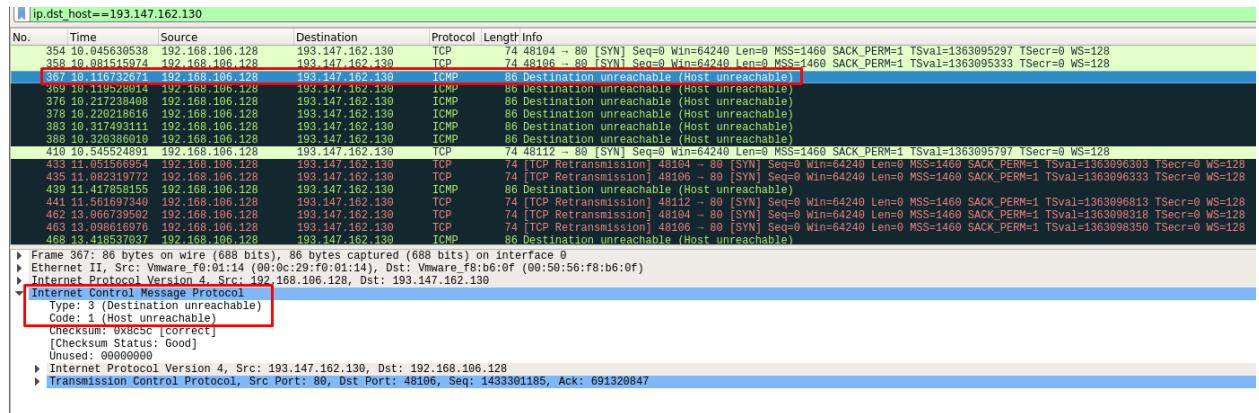


Figura 3.11 - Resultado ejemplo reject

Como se puede ver en la Figura 3.11, se envían peticiones a la web de trajano, la cual, al ser un servidor HTTP recibirá estas en el puerto 80, cuando se recibe la respuesta del servidor, se descarta y se envía un mensaje ICMP de tipo 3 (*Destination Unreachable*) de vuelta al servidor de trajano.

Camino de los paquetes en el ejemplo de *reject*.

- 1) Al usar curl, el kernel genera un paquete TCP SYN con dirección destino, la Ip de Trajano.
- 2) El paquete cruza por los hooks de *output* y *postrouting* y sale del equipo.
- 3) Por *ingress* llega la respuesta del servidor de Trajano con el puerto origen 80.
- 4) El paquete cruza por *prerouting* y tras este se toma la decisión de *routing*, determinando que ese paquete se dirige a un proceso local y siguiendo su camino por el *hook input*.
- 5) Al llegar a *input*, se hace *match* con la regla que este contiene y se realiza el *reject* del paquete
- 6) El kernel genera un paquete ICMP, con destino, la dirección del servidor de trajano.
- 7) Este paquete llega a *output* y se le aplica la regla con la acción *jump* *cadena_usuario*
- 8) El paquete se contabiliza en la regla de *counter* y al no hacer *match* con ninguna de las otras reglas, se devuelve a la cadena *output* y se aplica la política por defecto(*accept*)
- 9) El paquete cruza por *postrouting* y sale del equipo.

Conclusión del ejemplo:

El uso de la acción *reject*, es usado sobre todo para tratar el tráfico de las redes internas, ya que como hemos visto, este envía un mensaje ICMP, indicándole al usuario que no se ha podido completar la petición realizada.

Si esto mismo se hiciera para la red externa, estaríamos dando información al atacante, que normalmente no desearíamos dar, en este caso es preferible el uso de la acción *drop*.

Tipos de respuesta del reject:

Se puede elegir entre distintos tipos de respuestas ICMP, a continuación, se proponen los diferentes tipos y los comandos para poder crear las reglas:

- **net-unreachable:** Destination network unreachable

```
$ nft add rule inet filter input tcp sport 80 reject with icmp type net-unreachable
```

- **host-unreachable:** Destination host unreachable

```
$ nft add rule inet filter input tcp sport 80 reject with icmp type host-unreachable
```

- **prot-unreachable:** Destination protocol unreachable

```
$ nft add rule inet filter input tcp sport 80 reject with icmp type prot-unreachable
```

- **port-unreachable:** Destination port unreachable (this is the default)

```
$ nft add rule inet filter input tcp sport 80 reject with icmp type port-unreachable
```

- **net-prohibited:** Network administratively prohibited

```
$ nft add rule inet filter input tcp sport 80 reject with icmp type net-prohibited
```

- **host-prohibited:** Host administratively prohibited

```
$ nft add rule inet filter input tcp sport 80 reject with icmp type host-prohibited
```

- **admin-prohibited:** Communication administratively prohibited

```
$ nft add rule inet filter input tcp sport 80 reject with icmp type admin-prohibited
```

3.2.3.2.2 Acción Return, Accept y Drop

En el siguiente ejemplo, se verá la acción terminal *return*, la cual, puede ser usada de dos formas distintas:

- Si se invoca desde una cadena de usuario, en el momento en el que se llega a una regla con *return*, se para de procesar las reglas en esta y se vuelve a procesar las reglas de la cadena base desde la que se dio el salto y desde la regla en la que se dejó.
- Si se invoca desde una cadena base, al no poder volver a una cadena anterior puesto que no es una cadena de usuario, se toma la acción como un *accept*.

Ejemplo return, accept y drop.

Descripción del Ejemplo:

En este ejemplo, se verá el uso de la acción *return* y como esta, ante un mismo tráfico, procesa unas reglas y otras no, como consecuencia de esta acción.

Para ello, se crearán dos reglas a las cuales el tráfico pueda hacer *match* en la cadena *usuario*, la primera tendrá como acción *return*, mientras que la segunda tendrá *accept* y en la cadena desde la que salten, estará otra regla en la cual también puede hacerse *match* y cuya acción será *drop*.

En este ejemplo se partirá de la estructura creada en “Cadena de usuario”.

Análisis del ejemplo:

1. [Equipo:Encaminador - Usuario:root] Se añaden las siguientes reglas a la estructura de nftables:

- 1) La primera regla se añade a la cadena *usuario* , de forma que hace *match* si la dirección destino es la 2.2.2.2, se contabiliza y su acción es *return*.
- 2) La segunda regla se añade a la cadena *usuario* , de forma que hace *match* si la dirección destino es la 2.2.2.2, se contabiliza y su acción es *accept*.
- 3) La tercera regla se añade a la cadena *output*, de forma que hace *match* si la dirección destino es la 2.2.2.2, se contabiliza y su acción es *drop*.

Se puede ver en la Figura 3.12, como queda la estructura de nftables.

```
Equipo:Encaminador Usuario:root

$ nft add rule inet filter cadena_usuario ip daddr 2.2.2.2/32
counter return
$ nft add rule inet filter cadena_usuario ip daddr 2.2.2.2/32
counter accept
$ nft add rule inet filter output ip daddr 2.2.2.2/32 counter drop
```

The screenshot shows a terminal window with the title 'root@linux: /home/dit'. The window displays the following nftables configuration:

```
root@linux: /home/dit
Archivo Editar Ver Buscar Terminal Ayuda
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
        tcp sport http reject with icmp type host-unreachable # handle 16
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
        ip protocol icmp jump cadena_usuario # handle 6
        ip daddr 2.2.2.2 counter packets 0 bytes 0 drop # handle 21
    }

    chain cadena_usuario {
        counter packets 5424 bytes 390963 # handle 8
        ip daddr 8.8.8.8 counter packets 3 bytes 252 drop # handle 9
        ip daddr 1.1.1.1 counter packets 9 bytes 756 accept # handle 10
        ip daddr 2.2.2.2 counter packets 0 bytes 0 return # handle 24
        ip daddr 2.2.2.2 counter packets 0 bytes 0 accept # handle 25
    }
}
root@linux:/home/dit#
```

Figura 3.12 - Ejemplo return, accept y drop

2. [Equipo:Encaminador - Usuario:root] Se realiza un envío ICMP a la dirección Ip: 2.2.2.2/32 en el terminal 2 y se comprueba si este es capaz de salir y el valor de los contadores en el terminal 1.

- Terminal 1; Se muestra el *ruleset* de nftables.

```
Equipo: Encaminador Usuario: root
```

```
$ nft list ruleset
```

- Terminal 2; Se realiza el ping a la dirección Ip 2.2.2.2

```
Equipo: Encaminador Usuario: dit
```

```
$ ping -c 3 2.2.2.2
```

The screenshot shows two terminal windows side-by-side. The left window, titled 'root@linux:/home/dit', displays the contents of the /etc/nftables.conf file. It includes several chains: input, forward, output, and cadena_usuario, each with specific rules involving ICMP and TCP protocols. The right window, titled 'dit@linux: ~', shows the command 'ping -c 3 2.2.2.2' being run. The output shows three packets transmitted, but no received, indicating a 100% packet loss. The terminal also shows statistics for the ping and ends with a '^C' interrupt.

```
root@linux:/home/dit
Archivo Editar Ver Buscar Terminal Ayuda
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
        tcp sport http reject with icmp type host-unreachable # handle 16
    }
    chain forward {
        type filter hook forward priority 0; policy accept;
    }
    chain output {
        type filter hook output priority 0; policy accept;
        ip protocol icmp jump cadena_usuario # handle 6
        ip daddr 2.2.2.2 counter packets 3 bytes 252 drop # handle 21
    }
    chain cadena_usuario {
        counter packets 5525 bytes 398271 # handle 8
        ip daddr 8.8.8.8 counter packets 3 bytes 252 drop # handle 9
        ip daddr 1.1.1.1 counter packets 9 bytes 756 accept # handle 10
        ip daddr 2.2.2.2 counter packets 3 bytes 252 return # handle 24
        ip daddr 2.2.2.2 counter packets 0 bytes 0 accept # handle 25
    }
}
root@Linux:/home/dit#
```

```
dit@linux:~$ ping 2.2.2.2
PING 2.2.2.2 (2.2.2.2) 56(84) bytes of data.
ping: sendmsg: Operación no permitida
ping: sendmsg: Operación no permitida
ping: sendmsg: Operación no permitida
^C
--- 2.2.2.2 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2049ms
dit@linux:~$
```

Figura 3.13 - Resultado ejemplo return, accept y drop

Como se puede ver en la Figura 3.13, el ping es incapaz de completarse y el contador de la regla que sigue a *return* permanece en 0, viendo como así la regla no continua su evaluación en la cadena de usuario.

Camino del paquete ICMP generado en el ejemplo:

- El paquete ICMP es generado por el kernel, se toma la decisión de routing (destino 2.2.2.2) del mismo y este llega a la cadena *output* de la tabla filter.
- El paquete hace *match* con la regla de *output* ya que el protocolo de este es ICMP, por lo que su acción es *jump* a la cadena *_usuario*
- En la cadena *_usuario* aumenta en 1 el contador de la primera regla y hace *match* con la cuarta regla, aumentando el contador de esta y tomando la acción de *return* sobre el paquete, de forma que este vuelve a la cadena desde la que saltó (*output*).
- El paquete continua su camino con la siguiente regla a la del *jump*, de forma que hace *match* con la segunda regla y se toma la decisión de *drop* sobre el paquete.

Conclusión del ejemplo:

Una vez visto el ejemplo de uso de *return* y de las cadenas de usuario, podemos ver que trabajan muy bien en combinación, ya que podemos cortar el procesado de un paquete en una cadena de usuario, para darle un tratamiento más específico o más genérico en la cadena base.

3.2.3.3 Acciones No Terminales

En el punto anterior, se ha visto como son las acciones terminales, que son aquellas que ponen fin a la evaluación de una regla, las no terminales, son aquellas que toman una acción permitiendo que el paquete siga su evaluación por el resto de reglas que componen la cadena.

Los distintos tipos de acciones no terminales se muestran en la Tabla 6:

Acción No Terminal	Descripción
Log	Esta acción nos permite guardar un log del paquete que atraviesa esa regla en el sistema.
Counter	Esta acción nos permite contabilizar los paquetes y el tamaño de estos al coincidir con la regla. Tenemos que tener cuidado con esta acción terminal ya que es importante saber en qué punto de la regla se introduce, puesto que, al evaluarse la regla de izquierda a derecha, puede variar nuestro contador dependiendo de donde lo coloquemos, es por ello que de manera general se recomienda el uso de esta acción, antecediendo a una acción terminal.
Snat	Realiza una traducción NAT de nuestro paquete, en este caso un NAT de la dirección origen (Módulo NAT)
Masquerade	Al igual que el anterior, realiza un NAT de la dirección origen, pero de manera especial. Y es que esta toma como nueva dirección origen aquella que tenga la interfaz por la que sale. (Módulo NAT)
Dnat	Realiza una traducción NAT de nuestro paquete, en este caso es de una dirección destino. (Módulo NAT)
Redirect	Al igual que el anterior, este realiza un NAT de destino, pero de forma especial. Ya que al usar <i>redirect</i> lo que hacemos es que la dirección destino del paquete sea aquella que tiene nuestra máquina. (Módulo NAT)
Add/Update	Sirve para añadir o actualizar elementos de un conjunto.

Tabla 6 - Acciones no terminales

De las acciones vistas en la Tabla 6, aquellas que tienen en su descripción (Módulo NAT), se verán en el punto de “Traducción de direcciones (NAT)”.

A continuación, se pasará a ver un ejemplo de acción no terminal, como es *log*.

3.2.3.3.1 Acción Log

Esta acción sirve para que se pueda dejar constancia en el sistema, de un determinado tipo de paquete o flujo al cuál queramos prestarle especial atención.

Ejemplo acción log.

Descripción del Ejemplo:

Para ver el funcionamiento de la acción *log*, se plantea una regla muy sencilla, la cual estará situada en la cadena *input* y que además de realizar el *log* del paquete, lleve un contador del mismo.

En este ejemplo, se parte de la estructura creada en el punto “Cadena de usuario”.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Se crea la siguiente regla, la cual realizará el *log* de todos los paquetes cuyo destino sea un proceso local.

```
Equipo:Encaminador Usuario:root
```

```
$ nft add rule inet filter input counter log prefix \"New log: \"  
accept
```

La acción terminal *log* tiene dos opciones, *prefix* y *level*:

- *prefix*, sirve para indicar el prefijo con el que se desea que se guarde el *log*.
- *level*, sirve para indicar la importancia del *log*, sus valores pueden ser; *debug, alert, emerg, crit, info, err, warn o notice*.

2. **[Equipo:Encaminador - Usuario:dit]** Se realiza una petición TCP a la dirección Ip: 34.107.221.82

```
Equipo:Encaminador Usuario:dit
```

```
$ curl 34.107.221.82
```

3. **[Equipo:Encaminador - Usuario:root]** Se abre un terminal y se visualiza el *ruleset* de nftables.

En la Figura 3.14 se puede ver como queda la estructura de las reglas.

```
root@linux: /home/dit
Archivo Editar Ver Buscar Terminal Ayuda
root@linux:/home/dit# nft list ruleset -a
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
        counter packets 0 bytes 0 log prefix "New Log: " accept # handle 14
    }
    chain forward {
        type filter hook forward priority 0; policy accept;
    }
    chain output {
        type filter hook output priority 0; policy accept;
        ip protocol icmp jump cadena_usuario # handle 6
    }
    chain cadena_usuario {
        counter packets 20 bytes 1871 # handle 8
        ip daddr 8.8.8.8 counter packets 3 bytes 252 drop # handle 9
        ip daddr 1.1.1.1 counter packets 9 bytes 756 accept # handle 10
    }
}
root@linux:/home/dit#
```

Figura 3.14 - Ejemplo acción log

4. [Equipo:Encaminador - Usuario:root] Se procede a ver los *logs* creados en el sistema en otro terminal.

- Terminal 1; Se visualizan las reglas de nftables.
- Terminal 2; Se visualizan los *logs* generados en el sistema por el tráfico.

Para poder ver los *logs* del sistema se introduce el siguiente comando.

Equipo:Encaminador Usuario:root

```
$ cat /var/log/kern.log | grep New
```

De esta manera, se accede a la ubicación donde se guardan los *logs* del sistema y se realiza una búsqueda de la palabra *New* ya que es el prefijo que se ha decidido poner a los *logs* de los paquetes que atraviesen la cadena *input*.

```
root@linux:/home/dit# nft list ruleset -a
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
        counter packets 4 bytes 160 log prefix "New Log: " accept # handle 14
    }
    chain forward {
        type filter hook forward priority 0; policy accept;
    }
    chain output {
        type filter hook output priority 0; policy accept;
        ip protocol icmp jump cadena_usuario # handle 6
    }
    chain cadena_usuario {
        counter packets 20 bytes 1871 # handle 8
        ip daddr 8.8.8.8 counter packets 3 bytes 252 drop # handle 9
        ip daddr 1.1.1.1 counter packets 9 bytes 756 accept # handle 10
    }
}
root@linux:/home/dit#
```



```
root@linux:/home/dit#
Archivo Editar Ver Buscar Terminal Ayuda
rtt min/avg/max/mdev = 5.205/13.672/29.366/11.109 ms
dit@linux:~$ cat /var/log/kern.log | grep New
cat: /var/log/kern.log: Permiso denegado
dit@linux:~$ su root
Contraseña:
no session address file found in /root/.dbus/session-bus
root@linux:/home/dit# cat /var/log/kern.log | grep New
Jan 11 17:09:14 linux kernel: [10896.688967] New Log: IN=ens33 OUT= MAC=00:0c:29:f0:01:14:00:50:56:f8:b6:0f:08:00 SRC=34.107.221.82 DST=192.168.106.128 LEN=40 TOS=0x00 PREC=0x00 TTL=128 ID=61027 PROTO=TCP SPT=80 DPT=40148 WINDOW=64239 RES=0 x00 ACK URGP=0
Jan 11 17:09:14 linux kernel: [10896.689210] New Log: IN=ens33 OUT= MAC=00:0c:29:f0:01:14:00:50:56:f8:b6:0f:08:00 SRC=34.107.221.82 DST=192.168.106.128 LEN=40 TOS=0x00 PREC=0x00 TTL=128 ID=61028 PROTO=TCP SPT=80 DPT=40146 WINDOW=64239 RES=0 x00 ACK URGP=0
Jan 11 17:09:14 linux kernel: [10896.732271] New Log: IN=ens33 OUT= MAC=00:0c:29:f0:01:14:00:50:56:f8:b6:0f:08:00 SRC=34.107.221.82 DST=192.168.106.128 LEN=40 TOS=0x00 PREC=0x00 TTL=128 ID=61029 PROTO=TCP SPT=80 DPT=40148 WINDOW=64239 RES=0 x00 ACK PSH FIN URGP=0
Jan 11 17:09:14 linux kernel: [10896.732428] New Log: IN=ens33 OUT= MAC=00:0c:29:f0:01:14:00:50:56:f8:b6:0f:08:00 SRC=34.107.221.82 DST=192.168.106.128 LEN=40 TOS=0x00 PREC=0x00 TTL=128 ID=61030 PROTO=TCP SPT=80 DPT=40146 WINDOW=64239 RES=0 x00 ACK PSH FIN URGP=0
root@linux:/home/dit# cat /var/log/kern.log | grep New
```

Figura 3.15 - Resultado del ejemplo log

Como se puede ver en Figura 3.15, en el terminal 2, la regla ha registrado 4 paquetes que son exactamente los mismos que aparecen en el contador de la regla de la cadena *input*.

Conclusión del ejemplo:

Esta acción suele ser usada en entornos reales, para dejar constancia del tráfico que cumpla ciertas características límite, como accesos a puertos desde puntos de la red donde no se debería intentar acceder, equipos con un ratio de paq/segundo sospechoso etc...

Log, es la acción más usada para dejar constancia de ciertos paquetes, su uso suele estar ligado a el uso de reglas muy específicas, por ello se suelen usar sobre todo en cadenas de usuario.

3.2.3.3.2 Acción Counter

Ejemplo acción counter.

Descripción del Ejemplo:

En este apartado, se mostrará por qué se ha de tener cuidado con la acción *counter* y la importancia de su buen posicionamiento dentro de una regla.

Para realizar este ejemplo, se añadirán dos reglas a la cadena *output*, con la posición de la opción *counter* en diferentes puntos y se verá el resultado que tiene a la hora de procesar una regla.

Análisis del ejemplo:

1. [Equipo:Encaminador - Usuario:root] Se crean las siguientes reglas en nftables.
 - 1) La primera de ellas tendrá la acción *counter* tras indicar la cadena donde se añadirá la regla.
 - 2) La segunda de ellas tendrá la acción *counter* justo antes de la acción terminal *accept*.

Equipo:Encaminador Usuario:root

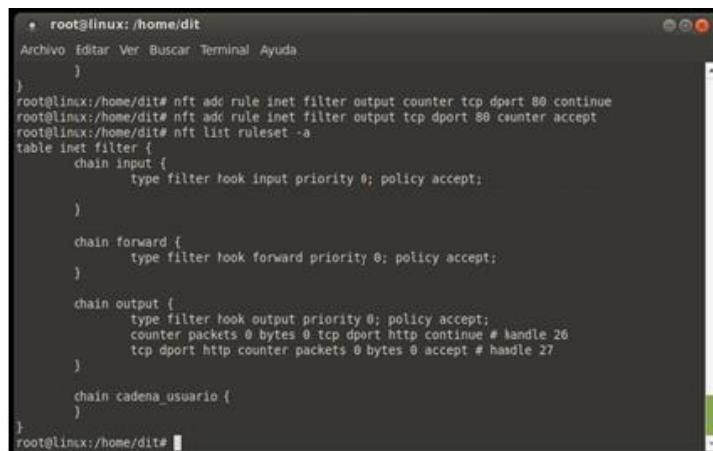
```
$ nft add rule inet filter output counter tcp dport 80 continue  
$ nft add rule inet filter output tcp dport 80 counter accept
```

2. [Equipo:Encaminador - Usuario:root] Se muestran las reglas creadas en nftables.

Equipo:Encaminador Usuario:root

```
$ nft list ruleset
```

Las reglas creadas pueden verse en la Figura 3.16.



A screenshot of a terminal window titled 'root@linux:/home/dit#'. The window shows the command 'nft list ruleset' being run. The output displays the two nftables rules created earlier. The first rule adds a counter to the 'output' chain of the 'inet' table. The second rule adds an 'accept' action to the 'output' chain, preceded by a 'counter' action. The terminal window has a dark background with white text and standard Linux window controls at the top.

```
* root@linux:/home/dit#  
Archivo Editar Ver Buscar Terminal Ayuda  
)  
)  
root@linux:/home/dit# nft add rule inet filter output counter tcp dport 80 continue  
root@linux:/home/dit# nft add rule inet filter output tcp dport 80 counter accept  
root@linux:/home/dit# nft list ruleset -a  
table inet filter {  
    chain input {  
        type filter hook input priority 0; policy accept;  
    }  
    chain forward {  
        type filter hook forward priority 0; policy accept;  
    }  
    chain output {  
        type filter hook output priority 0; policy accept;  
        counter packets 0 bytes 0 tcp dport http continue # handle 26  
        tcp dport http counter packets 0 bytes 0 accept # handle 27  
    }  
    chain cadena_usuario {  
    }  
}  
root@linux:/home/dit#
```

Figura 3.16 - Ejemplo posición counter

Con estas reglas, se quiere demostrar que la primera de ellas contabilizará todo el tráfico que curse por la cadena *output* ya sea TCP, UDP o ICMP, mientras que la segunda de las reglas, solo contabilizará aquel tráfico que sea TCP y cuyo puerto destino sea el 80.

3. [Equipo:Encaminador - Usuario:dit] En otro terminal, se realiza una petición al servidor trajano y un ping a Google.es

Equipo:Encaminador Usuario:dit

```
$ curl 193.147.162.130
$ ping -c 6 google.es
```

- Terminal 1; Se visualizan las reglas de nftables
- Terminal 2; Se realizan la petición a trajano y el ping.

The screenshot shows two terminal windows side-by-side. The left window, titled 'root@linux: /home/dit', displays the nftables configuration. It includes a chain 'cadena_usuario' and a table 'inet filter' with several chains: 'input', 'forward', 'output', and 'cadena_usuario'. The 'output' chain contains a rule for port 80. The right window, titled 'dit@linux: ~', shows the output of a 'ping google.es' command. The ping statistics show 6 packets transmitted, 6 received, 0% packet loss, and a round-trip time of 5011ms. The 'rtt min/avg/max/mdev' values are 16.060/28.373/53.304/15.155 ms.

```

root@linux: /home/dit
Archivo Editar Ver Buscar Terminal Ayuda
chain cadena_usuario {
}
root@linux:/home/dit# nft delete rule inet filter input handle 16
root@linux:/home/dit# nft list ruleset -a
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }
    chain forward {
        type filter hook forward priority 0; policy accept;
    }
    chain output {
        type filter hook output priority 0; policy accept;
        counter packets 624 bytes 62574 tcp dport http continue # handle 26
        tcp dport http counter packets 68 bytes 4080 accept # handle 27
    }
    chain cadena_usuario {
}
root@linux:/home/dit# 

dit@linux: ~
Archivo Editar Ver Buscar Terminal Ayuda
dit@linux:$ ping google.es
PING google.es (216.58.215.131) 56(84) bytes of data.
64 bytes from mad41s04-in-f3.1e100.net (216.58.215.131): icmp_seq=1 ttl=128 time
=53.3 ms
64 bytes from mad41s04-in-f3.1e100.net (216.58.215.131): icmp_seq=2 ttl=128 time
=20.2 ms
64 bytes from mad41s04-in-f3.1e100.net (216.58.215.131): icmp_seq=3 ttl=128 time
=45.6 ms
64 bytes from mad41s04-in-f3.1e100.net (216.58.215.131): icmp_seq=4 ttl=128 time
=16.0 ms
64 bytes from mad41s04-in-f3.1e100.net (216.58.215.131): icmp_seq=5 ttl=128 time
=16.9 ms
64 bytes from mad41s04-in-f3.1e100.net (216.58.215.131): icmp_seq=6 ttl=128 time
=17.9 ms
^C
--- google.es ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5011ms
rtt min/avg/max/mdev = 16.060/28.373/53.304/15.155 ms
dit@linux:$

```

Figura 3.17 - Resultado ejemplo posición counter

Como se puede observar en la Figura 3.17, la primera regla ha llevado la contabilidad de todo el tráfico que saliese por *output*, sin discriminar, mientras que la segunda solo ha realizado la cuenta de aquellos paquetes TCP que tuvieran como puerto destino el 80.

En el “Anexo G - CheatSheet Nftables”, se incluye un resumen de las opciones de las acciones, así como un cuadro recordatorio para trabajar con las reglas, donde se indica como crearlas, eliminarlas, insertarlas etc...

Del mismo modo, se encuentran cuadros resúmenes con las diferentes opciones de filtrado más usadas.

4 FLUJO DE PAQUETES (PACKET FLOW)

En este capítulo, se analizará el camino que sigue un paquete dentro del sistema de Netfilter hasta llegar a su destino.

Para ver este camino, se ha realizado un estudio del código fuente de Netfilter, de forma que se le oculta la mayor parte de la complejidad del código al lector y se ofrece una visión más amistosa del transcurso de los paquetes por Linux[25] y por los diferentes *hooks*[26] que proporciona Netfilter.

Para ver el camino que los paquetes hacen por las diferentes familias de tablas y el orden de los *hooks*, usaremos dos funcionalidades de nftables (*monitor* y *nftrace*).

4.1 Diagrama Flujo de paquetes

Nftrace nos permite marcar los paquetes que lleguen a una cadena y que sean de un tipo especificado por la regla.

La opción *monitor* se suele usar para ver los cambios en las reglas de nftables, en este caso, se usará para ver el camino que hace el paquete marcado por *nftrace* a través de cada una de las tablas y cadenas de Netfilter.

Para ello, primero necesitamos configurar el esquema básico de reglas (armazón) que se usará para guardar las reglas por las que atravesará el paquete[27].

Para orientarnos, en la Figura 4.1, se muestra el camino que sigue un paquete atravesando el sistema de *hooks* de Netfilter.

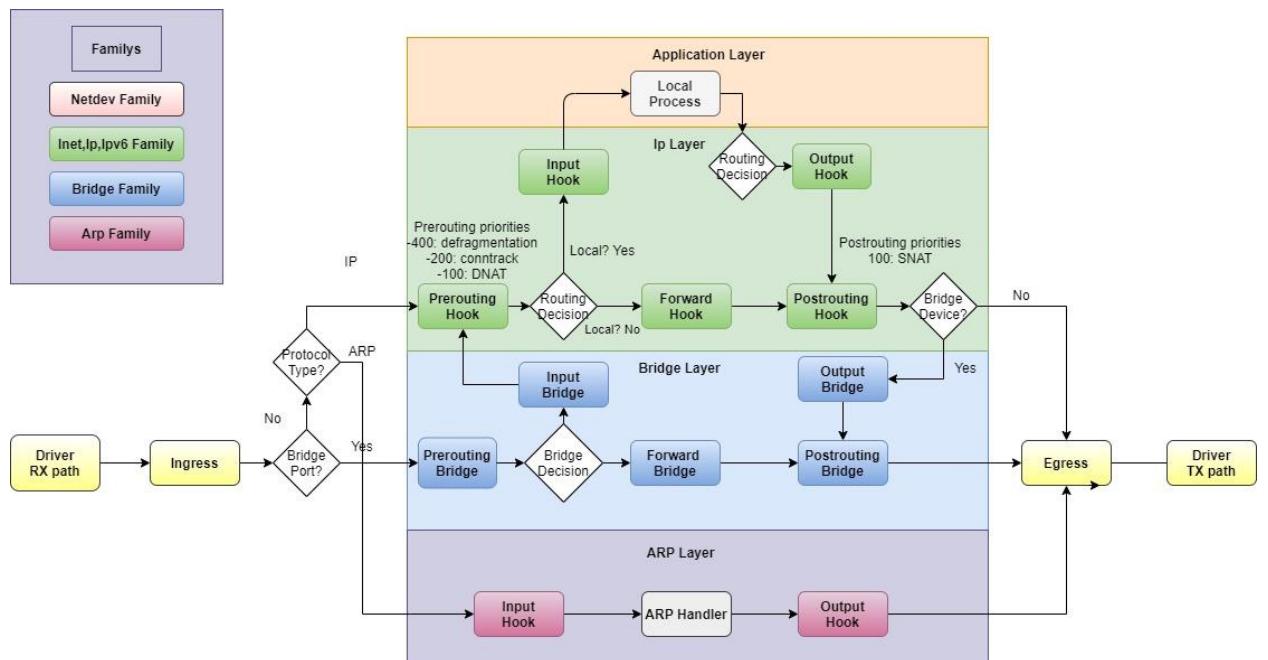


Figura 4.1 - Flow de paquetes de Nftables

Como se puede ver en la *Figura 4.1*, se ha dividido en 4 capas (*Arp Layer, Bridge Layer, Ip Layer y Application Layer*) a través de las cuales, definimos los *hooks* que actúan en cada una de ellas y el tipo de familia a la que pertenecen.

El camino que realiza un paquete, una vez que este llega y es tratado por el driver de nuestra NIC, es el indicado en la Figura 4.1.

En este camino, el primer *hook* que se encuentra el paquete es *ingress*, el cual, pertenece a la familia *netdev* y que al ser el primer *hook*, nos permite realizar un tratamiento temprano de los paquetes, así como conseguir realizar *bypass* de los mismos usando las *flowtable*.

Una vez el paquete atraviesa *ingress*, se plantea, si este ha entrado por el puerto de un *bridge* planteando una bifurcación de caminos.

- En caso de que la respuesta a esta pregunta, sea **SI**, se entregará el paquete al *hook prerouting* de la familia *bridge* (Camino Bridge)
- En caso contrario, se preguntará por el tipo de protocolo del paquete, según este sea ARP o Ip se entregará al *hook* correspondiente de su familia (Camino ARP o Camino Ip).

4.1.1 Camino Bridge

Si el paquete ha entrado por un puerto *bridge*, se entregará al *hook prerouting* de la familia *bridge* y tras atravesarlo se tomará la *bridge decision*, en la cual se decidirá si el paquete es para nuestro equipo (ver Figura 4.2).

- Si el paquete es para nuestro equipo, este es cursado por el *hook input* de *bridge*, el cual enlazará con el *hook prerouting* de la familia Ip.
- Si el paquete no es para nuestro equipo, seguirá su curso atravesando el *hook forward* de la familia *bridge* y por último el *hook postrouting* de la familia *bridge*, para finalmente, salir por *egress*.

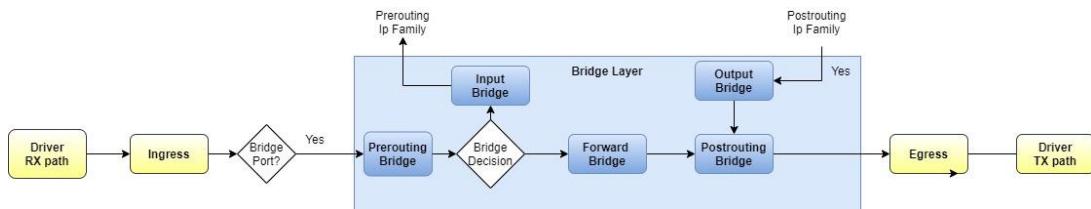


Figura 4.2 - Bridge Flow

4.1.2 Camino ARP

En caso de que en la decisión de *protocol type*, el protocolo sea ARP, el paquete seguirá su camino pasando por el *hook input* de la familia ARP hasta llegar al *Arp Handler* que será el encargado de procesar este paquete (Figura 4.3).

Si es nuestro equipo el que genera el paquete ARP, este nacerá del *Arp Handler* y cruzará a través del *hook output* de la familia ARP para salir finalmente por *egress*.

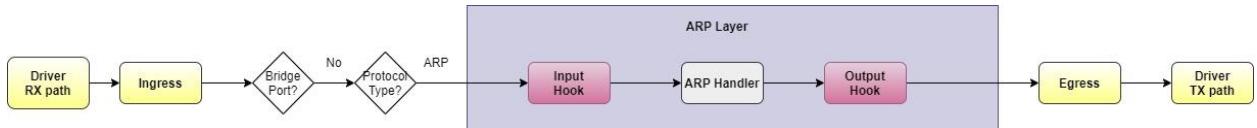


Figura 4.3 - ARP Flow

4.1.3 Camino Ip

En caso de que en la decisión de *protocol type*, el protocolo sea Ip, el paquete seguirá su curso atravesando el *hook prerouting* de la familia Ip.

Tras esto, se tomará la decisión de enrutamiento, para saber si el paquete es para un proceso local o necesita ser encaminado, en caso de que nuestro equipo permita el *forwarding*.

- Si el paquete es para un proceso local, tras la decisión de routing, este seguirá su curso por el *hook input* de la familia Ip, hasta entregarse al proceso local en la capa de aplicación.
- Si la decisión de enrutamiento, determina que el paquete no es para un proceso local y que el equipo permite el *forwarding*, este seguirá su curso por el *hook forward* y el *hook postrouting* de la familia Ip, llegado a ese punto, se tomará la decisión de si sale por una interfaz *bridge*.
 - Si la interfaz de salida es *bridge*, el paquete continuará su viaje por el *hook output* y el *hook postrouting* de la familia *bridge*, hasta que finalmente salga por *egress*.
 - Si la interfaz de salida no es *bridge*, el paquete continuará su camino y saldrá por *egress*.

Si el paquete es creado en un proceso local, este viajará hasta la capa Ip, en la cual se tomará la decisión de enrutamiento y tras esta, el paquete seguirá su camino por el *hook output* y el *hook postrouting* de la familia Ip, en este punto se tomará la misma decisión que en el camino anterior.

- Si la interfaz de salida es *bridge*, el paquete continuará su viaje por el *hook output* y el *hook postrouting* de la familia *bridge*, hasta que finalmente salga por *egress*.
- Si la interfaz de salida no es *bridge*, el paquete continuará su camino y saldrá por *egress*.

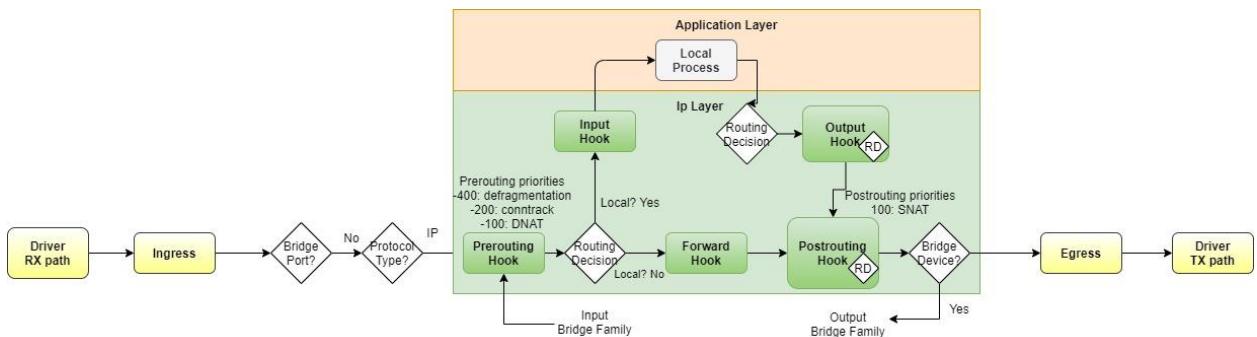


Figura 4.4 - Ip Flow

Como se puede ver en la Figura 4.4, en el *hook output* y en el *hook postrouting*, aparece un rombo con las siglas RD (Routing Decision) y es que, en estos *hooks*, ante el uso de reglas de tipo *NAT* o *route*, se realiza una reevaluación de la ruta, consultando de nuevo la tabla de routing, para ver si los cambios provocados por las reglas anteriores implican un cambio de ruta o de interfaz de salida.

A continuación, se realizarán pruebas usando el escenario de dos equipos.

4.2 Escenario pruebas flow paquetes.

Como se ha dicho, para este punto se usará el escenario de dos equipos, cuya configuración se encuentra en el “Anexo E - Escenario 2 Equipos”.

4.2.1 Ejemplo flow paquetes.

Descripción del ejemplo:

PC1 se encargará de generar el tráfico ICMP, que servirá para analizar el camino del paquete dentro del sistema de *hooks*.

El equipo Encaminador, configurará las reglas de nftables, para poder realizar el seguimiento del paquete. Además, este debe usar dos terminales, uno desde el que se configurarán las reglas y otro para realizar la monitorización del paquete.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Se crea en Encaminador el esquema de tablas y cadenas que se usará para hacer las pruebas.

Se crea la estructura de la tabla filter, que será la que contenga los *hooks* pertenecientes a la familia *inet* (*prerouting, input, forward, output* y *postrouting*).

```
Equipo:Encaminador Usuario:root

$ nft add table inet filter
$ nft add 'chain inet filter prerouting { type filter hook
prerouting priority -1 ; }'
$ nft add 'chain inet filter postrouting { type filter hook
postrouting priority 90 ; }'
$ nft add 'chain inet filter input { type filter hook input
priority 0 ; }'
$ nft add 'chain inet filter output { type filter hook output
priority 0 ; }'
$ nft add 'chain inet filter forward { type filter hook forward
priority 0 ; }'
```

2. **[Equipo:Encaminador - Usuario:root]** Se crea la tabla que contendrá los dos *hooks* de la familia ARP.

```
Equipo:Encaminador Usuario:root

$ nft add table arp filterarp
$ nft add 'chain arp filterarp inputarp { type filter hook input
priority 0 ; }'
$ nft add 'chain arp filterarp outputarp { type filter hook output
priority 0 ; }'
```

3. **[Equipo:Encaminador - Usuario:root]** Se crea la tabla que contendrá el *hook* de la familia *netdev* (*ingress*), como se puede ver, para poder crear la cadena para este *hook* es necesario que se especifique la interfaz en la cual escuchará el tráfico.

Equipo:Encaminador **Usuario:**root

```
$ nft add table netdev filteringress  
$ nft add 'chain netdev filteringress ingress { type filter hook  
ingress device ens38 priority 0 ; }'
```

4. [Equipo:Encaminador - Usuario:root] Se comprueba que se hayan creado correctamente las reglas, imprimiendo el *ruleset* de nftables, como se puede ver en la Figura 4.5.

Equipo:Encaminador **Usuario:**root

```
$ nft list ruleset
```

```
root@localhost:/home/dit# nft list ruleset  
table ip nat {  
    chain prerouting {  
        type nat hook prerouting priority 0; policy accept;  
        ip daddr 192.168.106.136 dnat to 10.10.1.20  
    }  
  
    chain postrouting {  
        type nat hook postrouting priority 100; policy accept;  
        ip saddr 10.10.1.20 counter packets 28 bytes 1752 masquerade  
    }  
}  
table inet filter {  
    chain prerouting {  
        type filter hook prerouting priority -1; policy accept;  
    }  
  
    chain postrouting {  
        type filter hook postrouting priority 90; policy accept;  
    }  
  
    chain input {  
        type filter hook input priority 0; policy accept;  
    }  
  
    chain output {  
        type filter hook output priority 0; policy accept;  
    }  
  
    chain forward {  
        type filter hook forward priority 0; policy accept;  
    }  
}  
table arp filterarp {  
    chain inputarp {  
        type filter hook input priority 0; policy accept;  
    }  
  
    chain outputarp {  
        type filter hook output priority 0; policy accept;  
    }  
}  
table netdev filteringress {  
    chain ingress {  
        type filter hook ingress device ens38 priority 0; policy accept;  
    }  
}  
root@localhost:/home/dit#
```

Figura 4.5 - Armazón hooks escenario flow

5. [Equipo:Encaminador - Usuario:root] A continuación se procede a crear las reglas.

- 1) Para realizar este seguimiento de los diferentes paquetes, se debe activar la marca *nftrace*, tanto en la cadena de *output* como en la de *ingress*.

Por ello, se crea una regla en la tabla *filteringress*, la cual se activará si el protocolo del paquete es ICMP.

Del mismo modo, se activa esta marca en el *hook output* de la familia Ip, para poder realizar el seguimiento de los paquetes que sean generados en un proceso local.

- 2) Para poder realizar un correcto seguimiento del camino de los paquetes, se realizará en cada regla dos acciones no terminales.

- Un *log*, con un prefijo que indique por qué regla se ha pasado.
- Un *counter*, que lleve la cuenta de los paquetes que atraviesan esa regla.

Equipo: Encaminador **Usuario:** root

```
#Se activa la marca nftrace en ingress y output

$ nft add rule netdev filteringress ingress meta nftrace set 1
$ nft add rule inet' filter output ip protocol icmp meta nftrace
set 1'

#Se añaden las reglas que harán los logs en las cadenas arp

$ nft add rule arp filterarp' inputarp log prefix "Entra ARP: "
counter accept'
$ nft add rule arp' filterarp outputarp meta nftrace set 1 '
$ nft add rule arp filterarp' outputarp log prefix "Sale ARP: "
counter accept'

#Se añade las reglas que harán los logs en las cadenas de filter

$ nft add rule inet' filter output ip protocol icmp log prefix
"Pasa por output-inet: " counter accept '
$ nft add rule inet' filter input ip protocol icmp log prefix
"Pasa por input-inet: " counter accept '
$ nft add rule inet' filter prerouting ip protocol icmp log prefix
"Pasa por pre-inet: " counter accept '
$ nft add rule inet' filter postrouting ip protocol icmp log prefix
"Pasa por post-inet: " counter accept '
$ nft add rule inet' filter forward ip protocol icmp log prefix
"Pasa por forward-inet: " counter accept '
```

6. **[Equipo:Encaminador - Usuario:root]** Se muestra el *ruleset* de nftables, para comprobar que se han creado correctamente.

En un segundo terminal, se activa el monitor de *nftrace*.

En la Figura 4.6, se puede comprobar, como debe quedar la estructura de reglas y cadenas.

Equipo: Encaminador **Usuario:** root

```
#Terminal 1
$ nft list ruleset

#Terminal 2
$nft monitor trace
```

```

table inet filter {
    chain prerouting {
        type filter hook prerouting priority -1; policy accept;
        ip protocol icmp log prefix "Pasa por pre-inet: " counter packets 0 bytes 0 accept # handle 23
    }

    chain postrouting {
        type filter hook postrouting priority 90; policy accept;
        ip protocol icmp log prefix "Pasa por post-inet: " counter packets 0 bytes 0 accept # handle 24
    }

    chain input {
        type filter hook input priority 0; policy accept;
        ip protocol icmp log prefix "Pasa por input-inet: " counter packets 0 bytes 0 accept # handle 22
    }

    chain output {
        type filter hook output priority 0; policy accept;
        ip protocol icmp nftrace set 1 # handle 20
        ip protocol icmp log prefix "Pasa por output-inet: " counter packets 0 bytes 0 accept # handle 21
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
        ip protocol icmp log prefix "Pasa por forward-inet: " counter packets 0 bytes 0 accept # handle 25
    }
}
table arp filterarp {
    chain inputarp {
        type filter hook input priority 0; policy accept;
        log prefix "Entra ARP: " counter packets 0 bytes 0 accept # handle 3
    }

    chain outputarp {
        type filter hook output priority 0; policy accept;
        nftrace set 1 # handle 4
        log prefix "Sale ARP: " counter packets 0 bytes 0 accept # handle 5
    }
}
table netdev filteringress {
    chain ingress {
        type filter hook ingress device ens38 priority 0; policy accept;
        ip protocol icmp nftrace set 1 # handle 3
    }
}

```

root@localhost:/home/dit#

Figura 4.6 - Reglas escenario flow

7. [Equipo:PC1 - Usuario:root] Desde PC1, se realizan pings para comprobar el camino que siguen los paquetes.

- 1) El primero de ellos tendrá como destino un proceso local, por ello se realiza el ping a la Ip: 10.10.1.10,
- 2) El segundo, se usará para ver el *forwarding*, por ello se realiza el ping al DNS de Google, con Ip: 8.8.8.8.

La realización de los pings, puede comprobarse en la Figura 4.7.

Equipo:PC1 Usuario:root

```

$ ping -c 3 10.10.1.10
$ ping -c 3 8.8.8.8

```

```

[root@localhost dit]# ping 10.10.1.10
PING 10.10.1.10 (10.10.1.10) 56(84) bytes of data.
64 bytes from 10.10.1.10: icmp_seq=1 ttl=64 time=0.345 ms
64 bytes from 10.10.1.10: icmp_seq=2 ttl=64 time=0.277 ms
64 bytes from 10.10.1.10: icmp_seq=3 ttl=64 time=0.381 ms
^C
--- 10.10.1.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 13ms
rtt min/avg/max/mdev = 0.277/0.334/0.381/0.045 ms
[root@localhost dit]# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=127 time=75.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=127 time=12.10 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=127 time=65.3 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 6ms
rtt min/avg/max/mdev = 12.997/51.205/75.286/27.321 ms

```

Figura 4.7 - Prueba ICMP PC1 escenario flow

8. [Equipo:Encaminador - Usuario:root] Se procede a analizar los resultados de los *logs* y del monitor *ntrace*, del ping realizado al proceso local de Encaminador.

```
trace id 255d9669 arp filterarp outputarp rule ntrace set 1 (verdict continue)
trace id 255d9669 arp filterarp outputarp rule log prefix "Sale ARP: " counter packets 3 bytes 126 accept (verdict accept)
trace id 886d3e4b arp filterarp outputarp packet: oif "ens38" arp operation request
trace id 886d3e4b arp filterarp outputarp rule ntrace set 1 (verdict continue)
trace id 886d3e4b arp filterarp outputarp rule log prefix "Sale ARP: " counter packets 3 bytes 126 accept (verdict accept)
trace id a9956702 netdev filteringress ingress packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip
614 ip length 84 icmp type echo-request icmp code 0 icmp id 12246 icmp sequence 1
trace id a9956702 netdev filteringress ingress_rule_in_protocol_icmp ntrace set 1 (verdict continue)
trace id a9956702 netdev filteringress ingress verdict continue
trace id a9956702 netdev filteringress ingress
trace id a9956702 inet filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp
length 84 icmp type echo-request icmp code 0 icmp id 12246 icmp sequence 1
trace id a9956702 inet filter prerouting rule ip protocol icmp log prefix "Pasa por pre-inet: " counter packets 0 bytes 0 accept (verdict accept)
trace id a9956702 ip nat prerouting verdict continue
trace id a9956702 ip nat prerouting
trace id a9956702 inet filter input packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 i
th 84 icmp type echo-request icmp code 0 icmp id 12246 icmp sequence 1
trace id a9956702 inet filter input rule ip protocol icmp log prefix "Pasa por input-inet: " counter packets 0 bytes 0 accept (verdict accept)
```

Figura 4.8 - Nftrace ICMP local, entrada

En la Figura 4.8, se puede ver que *nftrace* crea un ID de seguimiento de paquete (a9956702) y que el primer *hook* por el que pasa es *ingress*, que es el encargado de activar su marca.

Camino del ICMP request hasta el proceso local:

- 1) El paquete ICMP req es recibido en la NIC de Encaminador y tras realizar las comprobaciones básicas de malformación, se entrega al *hook ingress*
- 2) En el *hook ingress*, se puede ver que se le aplica la regla de *nftrace* y se le asigna un identificador al paquete (a9956702), tras esto, se toma el veredicto de *continue* y se entrega al *hook prerouting*.
- 3) En el *hook prerouting*, se aplica la primera regla, la cual aumenta el contador de paquetes y guarda un *log* con el nombre “Pasa por pre-inet”, el veredicto del paquete es *continue*.
- 4) Se toma la decisión de *routing* sobre el paquete y se verifica que se dirige a un proceso local, por lo que este continua su camino por el *hook input*.
- 5) Al llegar al *hook input*, se aplica la primera regla, la cual aumenta el contador de esta en 1, y se guarda un *log* con el nombre “Pasa por input-inet”.
- 6) Tras esto se toma como veredicto *accept* y se entrega al proceso local.

Se puede observar además, que entra por la interfaz ens38 y que tiene ID ICMP (12246).

En la Figura 4.10, se pueden comprobar como las dos primeras entradas corresponden a los *logs* generados por los *hooks* de *prerouting* e *input*.

9. [Equipo:Encaminador - Usuario:root] Se procede a analizar los resultados de los *logs* y del monitor *ntrace*, de la respuesta al ping local generada por el equipo Encaminador.

```
trace id a9956702 inet filter input rule ip protocol icmp log prefix "Pasa por input-inet: " counter packets 0 bytes 0 accept (verdict accept)
trace id 7e137a80 inet filter output packet: oif "ens38" ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 25151 ip length 84 icmp type echo-reply icmp code 0 icmp id 12246 icmp sequence 1
trace id 7e137a80 inet filter output rule ip protocol icmp ntrace set 1 (verdict continue)
trace id 7e137a80 inet filter output rule ip protocol icmp log prefix "Pasa por output-inet: " counter packets 0 bytes 0 accept (verdict accept)
trace id 7e137a80 inet filter prerouting packet: oif "ens38" ip saddr 10.10.1.10 ip daddr 10.10.1.20 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 25151 ip length 84 icmp type echo-reply icmp code 0 icmp id 12246 icmp sequence 1
```

Figura 4.9 - Nftrace ICMP local Salida

En la Figura 4.9, se puede ver que se realiza un nuevo seguimiento de un paquete ICMP, en este caso generado por el proceso local con el ID (7e137a80).

Camino del ICMP response, hasta que sale de Encaminador:

- 1) El paquete ICMP res, es generado por el proceso local como respuesta al request recibido.
- 2) El paquete llega al *hook output* donde se comienza a hacer el seguimiento del mismo, se le asigna el ID (7e137a80).
- 3) Tras activar la traza de *nftrace* del paquete, se le aplica la segunda regla, aumentando el contador de esta y generando un *log* con el nombre “pasa por out-inet”, siendo su veredicto *continue*.
- 4) El paquete llega al *hook postrouting*, donde hace *match* con la primera regla y genera un *log* con el nombre “pasa por post-inet”, siendo su veredicto *accept*.
- 5) El paquete sale por la NIC “ens38” camino a PC1.

Se puede observar que el ID del paquete ICMP, es el mismo que el observado en la *Figura 4.8*, que seguía el camino de entrada, por lo que, se puede corroborar que este es la respuesta del paquete ICMP anterior.

10. [Equipo:Encaminador - Usuario:root] Se procede a analizar los *logs* del ping dirigido al proceso local del equipo Encaminador.

Equipo:Encaminador Usuario:root

```
$ cat /var/log/kern.log | grep Pasa
```

En la Figura 4.10, se puede ver, como las entradas 3 y 4 coinciden con los *logs* que se han mencionado en el camino del paquete ICMP response.

```
localhost kernel :[31925.271408] Pasa por pre-inet: IN=ens38 OUT=
MAC=00:0c:29:f0:01:1e:00:0c:29:62:ed:ad:08:00 SRC=10.10.1.20 DST=10.10.1.10 LEN=84 TOS=0x00
PREC=0x00 TTL=64 ID=39614 DF PROTO=ICMP TYPE=8 CODE=0 ID=12246 SEQ=1
localhost kernel :[31925.271433] Pasa por input-inet: IN=ens38 OUT=
MAC=00:0c:29:f0:01:1e:00:0c:29:62:ed:ad:08:00 SRC=10.10.1.20 DST=10.10.1.10 LEN=84 TOS=0x00
PREC=0x00 TTL=64 ID=39614 DF PROTO=ICMP TYPE=8 CODE=0 ID=12246 SEQ=1
localhost kernel: [31925.271455] Pasa por output-inet: IN= OUT=ens38 SRC=10.10.1.10 DST=10.10.1.20
LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=25151 PROTO=ICMP TYPE=0 CODE=0 ID=12246 SEQ=1
localhost kernel: [31925.271460] Pasa por post-inet: IN= OUT=ens38 SRC=10.10.1.10 DST=10.10.1.20
LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=25151 PROTO=ICMP TYPE=0 CODE=0 ID=12246 SEQ=1
```

Figura 4.10 - Logs del ICMP local al pasar por los hooks

La Figura 4.11, representa el camino del ping cuyo destino es un proceso local en Encaminador.

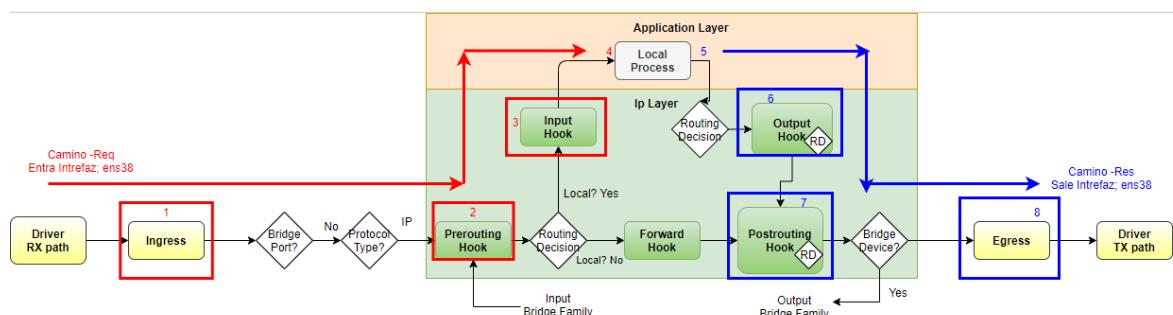


Figura 4.11 - Resumen camino ping a Encaminador

11. [Equipo:Encaminador - Usuario:root] Se procede a analizar los resultados del monitor *nftrace* del ping dirigido al DNS de Google.

```
root@localhost:/home/dmitri# nft monitor trace
trace id b9dc1dbc netdev filteringress ingress packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 8.8.8.8 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 35533
ip length 84 icmp type echo-request icmp code 0 icmp id 12304 icmp sequence 1
trace id b9dc1dbc netdev filteringress ingress rule ip protocol icmp nftrace set 1 (verdict continue)
trace id b9dc1dbc netdev filteringress ingress verdict continue
trace id b9dc1dbc netdev filteringress ingress
trace id b9dc1dbc netdev filteringress ingress rule ip protocol icmp log prefix "Pasa por pre-inet: " counter packets 3 bytes 252 accept (verdict accept)
trace id b9dc1dbc ip nat prerouting verdict continue
trace id b9dc1dbc inet filter prerouting rule ip protocol icmp log prefix "Pasa por pre-inet: " counter packets 3 bytes 252 accept (verdict accept)
trace id b9dc1dbc inet filter prerouting rule ip protocol icmp log prefix "Pasa por forward-inet: " counter packets 0 bytes 0 accept (verdict accept)
trace id b9dc1dbc inet filter postrouting packet: oif "ens33" ip saddr 10.10.1.20 ip daddr 8.8.8.8 ip dscp cs0 ip ecn not-ect ip ttl 63 ip id 35533 ip length 84 icmp type echo-request icmp code 0 icmp id 12304 icmp sequence 1
trace id b9dc1dbc inet filter postrouting rule ip protocol icmp log prefix "Pasa por post-inet: " counter packets 3 bytes 252 accept (verdict accept)
trace id b9dc1dbc ip nat postrouting packet: oif "ens33" ip saddr 10.10.1.20 ip daddr 8.8.8.8 ip dscp cs0 ip ecn not-ect ip ttl 63 ip id 35533 ip length 84 icmp type echo-request icmp code 0 icmp id 12304 icmp sequence 1
trace id b9dc1dbc ip nat postrouting rule ip saddr 10.10.1.20 counter packets 140 bytes 9397 masquerade (verdict accept)
```

Figura 4.12 - Nftrace ICMP externo entrada

En la Figura 4.12, se puede comprobar el transcurso del paquete ICMP por los distintos *hooks* en su camino de *forwarding*.

Camino del ICMP request hasta Google DNS:

- 1) El paquete ICMP req es recibido en la NIC de Encaminador y tras realizar las comprobaciones básicas de malformación, se entrega al *hook ingress*
- 2) En el *hook ingress*, se puede ver como se le aplica la regla de *nftrace* y se le asigna un identificador al paquete(b9dc1dbc), tras esto se toma el veredicto de *continue* y se entrega al *hook prerouting*.
- 3) En el *hook prerouting* se aplica la primera regla, la cual aumenta el contador de paquetes y guarda un *log* con el nombre “Pasa por pre-inet”, el veredicto del paquete es *continue*.
- 4) Se toma la decisión de *routing* sobre el paquete y se verifica que no es para un proceso local, por lo que este necesita ser encaminado, continuando así, su camino por el *hook forward*.
- 6) Al llegar al *hook forward*, se aplica la primera regla, la cual aumenta el contador de esta en 1, y se guarda un *log* con el nombre “Pasa por forward-inet” .
- 7) El paquete llega al *hook postrouting*, donde hace *match* con la primera regla, y genera un *log* con el nombre “Pasa por post-inet”, su veredicto es *accept*.
- 8) El paquete sale por la NIC “ens33” en dirección al DNS de Google.

Se comprueba que *nftrace* le da el ID (b9dc1dbc) a este paquete, que su interfaz de entrada es ens38 y que el ID del ICMP es (12304).

12. [Equipo:Encaminador - Usuario:root] Se procede a analizar los *logs* del ping dirigido al DNS de Google.

Equipo:Encaminador **Usuario:**root

```
$ cat /var/log/kern.log | grep Pasa
```

El camino de la respuesta es exactamente el mismo que el camino que ha recorrido el request, con la única salvedad de que request (entra por ens38 y sale por ens33) y res (entra por ens33 y sale por ens38).

Esto puede observarse en los *logs* generados en la Figura 4.13.

```

localhost kernel: [32151.121955] Pasa por pre-inet: IN=ens38 OUT=
MAC=00:0c:29:f0:01:1e:00:0c:29:62:ed:ad:08:00 SRC=10.10.1.20 DST=8.8.8.8 LEN=84 TOS=0x00
PREC=0x00 TTL=64 ID=35533 DF PROTO=ICMP TYPE=8 CODE=0 ID=12304 SEQ=1
localhost kernel: [32151.121984] Pasa por forward-inet: IN=ens38 OUT=ens33
MAC=00:0c:29:f0:01:1e:00:0c:29:62:ed:ad:08:00 SRC=10.10.1.20 DST=8.8.8.8 LEN=84 TOS=0x00
PREC=0x00 TTL=63 ID=35533 DF PROTO=ICMP TYPE=8 CODE=0 ID=12304 SEQ=1
localhost kernel: [32151.121990] Pasa por post-inet: IN= OUT=ens33 SRC=10.10.1.20 DST=8.8.8.8 LEN=84
TOS=0x00 PREC=0x00 TTL=63 ID=35533 DF PROTO=ICMP TYPE=8 CODE=0 ID=12304 SEQ=1
localhost kernel: [32151.196818] Pasa por pre-inet: IN=ens33 OUT=
MAC=00:0c:29:f0:01:14:00:50:56:f8:b6:0f:08:00 SRC=8.8.8.8 DST=192.168.106.136 LEN=84 TOS=0x00
PREC=0x00 TTL=128 ID=16445 PROTO=ICMP TYPE=0 CODE=0 ID=12304 SEQ=1
localhost kernel: [32151.196830] Pasa por forward-inet: IN=ens33 OUT=ens38
MAC=00:0c:29:f0:01:14:00:50:56:f8:b6:0f:08:00 SRC=8.8.8.8 DST=10.10.1.20 LEN=84 TOS=0x00
PREC=0x00 TTL=127 ID=16445 PROTO=ICMP TYPE=0 CODE=0 ID=12304 SEQ=1
localhost kernel: [32151.196832] Pasa por post-inet: IN= OUT=ens38 SRC=8.8.8.8 DST=10.10.1.20 LEN=84
TOS=0x00 PREC=0x00 TTL=127 ID=16445 PROTO=ICMP TYPE=0 CODE=0 ID=12304 SEQ=1

```

Figura 4.13 - Logs del ICMP externo al pasar por los hooks

Se debe distinguir entre los dos Id's que pueden tener los paquetes, tales como los vistos en la Figura 4.12.

Nftrace crea un ID para el paquete, que solo tiene sentido, entendido dentro de la estructura local de nftables de Encaminador, mientras que el segundo ID, es el propio del paquete ICMP y que viaja en la cabecera de este.

La Figura 4.14 , representa el camino del ping al DNS de Google, el cual, se encamina, puesto que su destino está en una máquina pública y no en el equipo Encaminador.

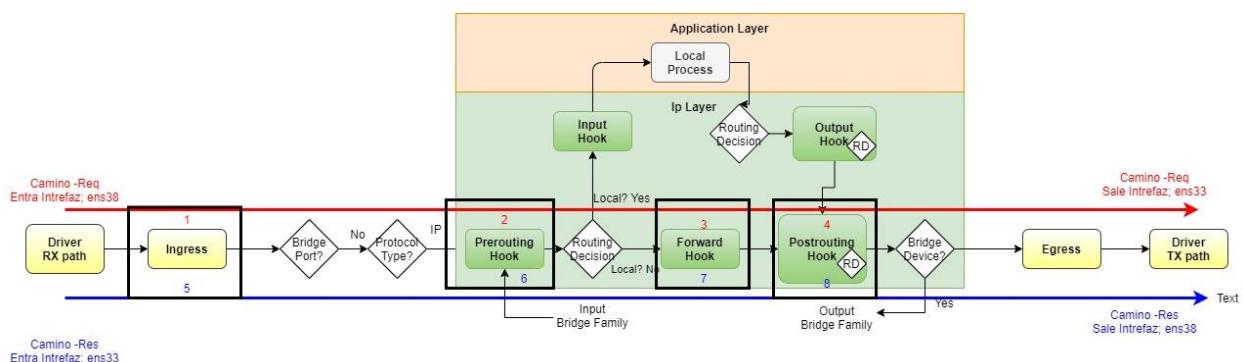


Figura 4.14 - Resumen camino ping a DNS Google

Conclusión del ejemplo:

Como se ha podido ver en los ejemplos, estos sirven para mostrar paso a paso el camino de los dos tipos de caminos que podemos encontrarnos en cualquier escenario:

- Paquetes dirigidos a nuestro equipo; Los cuales atraviesan los siguientes *hooks* (*ingress, prerouting, input, output y postrouting*).
- Paquetes que nuestro equipo debe encaminar; Los cuales atraviesan los siguientes *hooks* (*ingress, prerouting, forward y postrouting*).

4.3 Escenario prueba flow interfaz lo

En este punto, se estudiará el camino de los paquetes que viajan a través de la interfaz “lo”.

4.3.1 Ejemplo flow interfaz lo.

Descripción del ejemplo:

Como ejemplo particular, se comentará el flujo que sigue el tráfico que se genera en nuestro equipo y que viaja por la interfaz local, de forma, que se pueda comprender los *hooks* e interfaces que el paquete atraviesa en todo momento.

Para ello, se tomará el escenario del ejemplo anterior “Escenario pruebas flow paquetes.”, el cual, se basa en la configuración del “Anexo E - Escenario 2 Equipos”.

A este escenario, se le aplicarán ciertos cambios para poder adaptarlo a este nuevo ejemplo.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Se añade la siguiente regla al *ruleset* de Encaminador.

Como se quiere comprobar si el tráfico que va hacia la interfaz local deja en algún momento el equipo y por donde transita este, es preciso que se añada una cadena a la tabla *filteringress*, la cual esté asociada a la interfaz “*lo*”, ya que será en esta en la que se añadirá un contador para ver si el paquete entra y sale por esta interfaz.

```
Equipo:Encaminador Usuario:root
```

```
$ nft add chain netdev' filteringress in-lo { type filter hook
ingress device lo priority 0 ; }'
$ nft add rule netdev filteringress in-lo ip protocol icmp counter
$ nft list ruleset
```

2. **[Equipo:Encaminador - Usuario:root]** Se comprueba que las reglas creadas para esta prueba son las mostradas en la Figura 4.15, para ello, se usa el siguiente comando.

Además, se abre un nuevo terminal para activar el modo monitor.

```
Equipo:Encaminador Usuario:root
```

```
#Terminal 1
$ nft list ruleset

#Terminal 2
$ nft monitor trace
```

```

table inet filter {
    chain prerouting {
        type filter hook prerouting priority -1; policy accept;
        ip protocol icmp log prefix "Pasa por pre-inet: " counter packets 25 bytes 2100 accept
    }
    chain postrouting {
        type filter hook postrouting priority 90; policy accept;
        ip protocol icmp log prefix "Pasa por post-inet: " counter packets 25 bytes 2100 accept
    }
    chain input {
        type filter hook input priority 0; policy accept;
        ip protocol icmp log prefix "Pasa por input-inet: " counter packets 19 bytes 1596 accept
    }
    chain output {
        type filter hook output priority 0; policy accept;
        ip protocol icmp nftrace set 1
        ip protocol icmp log prefix "Pasa por output-inet: " counter packets 19 bytes 1596 accept
    }
    chain forward {
        type filter hook forward priority 0; policy accept;
        ip protocol icmp log prefix "Pasa por forward-inet: " counter packets 6 bytes 504 accept
    }
}
table arp filterarp {
    chain inputarp {
        type filter hook input priority 0; policy accept;
        log prefix "Entra ARP: " counter packets 499 bytes 22954 accept
    }
    chain outputarp {
        type filter hook output priority 0; policy accept;
        nftrace set 1
        log prefix "Sale ARP: " counter packets 427 bytes 17934 accept
    }
}
table netdev filteringress {
    chain ingress {
        type filter hook ingress device ens38 priority 0; policy accept;
        ip protocol icmp counter packets 0 bytes 0
    }
    chain in-lo {
        type filter hook ingress device lo priority 0; policy accept;
        ip protocol icmp counter packets 0 bytes 0
    }
}

```

Figura 4.15 - Reglas escenario flow interfaz lo

- [Equipo:Encaminador - Usuario:dit] Se abre Wireshark y se pone a escuchar en la interfaz “lo”, desde un nuevo terminal se realiza el envío del ping a la dirección local usando el siguiente comando.

Equipo:Encaminador Usuario:dit

```
# ping -c 1 127.0.0.1
```

- [Equipo:Encaminador - Usuario:root] Se comprueba el monitor de nft para ver el transcurso del paquete request.

```

root@localhost:/home/dit# nft monitor trace
trace id 246f157d inet filter output packet: oif "lo" ip saddr 127.0.0.1 ip daddr 127.0.0.1 ip dscp cs0 ip ecn not-ect ip ttl
pe echo-request icmp code 0 icmp id 4578 icmp sequence 1
trace id 246f157d inet filter output rule ip protocol icmp nftrace set 1 (verdict continue)
trace id 246f157d inet filter postrouting packet: oif "lo" ip saddr 127.0.0.1 ip daddr 127.0.0.1 ip dscp cs0 ip ecn not-ect ip
type echo-request icmp code 0 icmp id 4578 icmp sequence 1
trace id 246f157d inet filter postrouting rule ip protocol icmp log prefix "Pasa por post-inet: " counter packets 27 bytes 2268
trace id 246f157d ip nat postrouting verdict continue
trace id 246f157d ip nat postrouting
trace id 741c1ec7 netdev filteringress in-lo packet: iif "lo"
trace id 741c1ec7 netdev filteringress in-lo rule ip protocol icmp counter packets 2 bytes 168 (verdict continue)
trace id 741c1ec7 netdev filteringress in-lo verdict continue
trace id 741c1ec7 netdev filteringress in-lo
trace id 741c1ec7 inet filter prerouting packet: iif "lo" ip saddr 127.0.0.1 ip daddr 127.0.0.1 ip dscp cs0 ip ecn not-ect ip
type echo-request icmp code 0 icmp id 4578 icmp sequence 1
trace id 741c1ec7 inet filter prerouting rule ip protocol icmp log prefix "Pasa por pre-inet: " counter packets 27 bytes 2268
trace id 741c1ec7 ip nat prerouting verdict continue
trace id 741c1ec7 ip nat prerouting
trace id 741c1ec7 inet filter input packet: iif "lo" ip saddr 127.0.0.1 ip daddr 127.0.0.1 ip dscp cs0 ip ecn not-ect ip ttl
pe echo-request icmp code 0 icmp id 4578 icmp sequence 1

```

Figura 4.16 - Nftrace ICMP req interfaz lo

Como se puede observar en la Figura 4.16, cuando se lanza el ping, el primer *hook* por el que atraviesa es *output*, siendo su interfaz de salida “*lo*”.

Camino del ICMP request:

- 1) El paquete ICMP req, es generado por el proceso local.
- 2) El paquete llega al *hook output*, donde se comienza a hacer el seguimiento del mismo, se le asigna el identificador (246f157d).
- 3) Tras activar la traza de *nftrace* del paquete, a este se le aplica la segunda regla, aumentando el contador de esta y generando un *log* con el nombre “Pasa por output-inet”, su veredicto es *continue*.
- 4) El paquete llega al *hook postrouting*, donde hace *match* con la primera regla y genera un *log* con el nombre “Pasa por post-inet”, su veredicto es *continue*.
- 5) El paquete sale por la interfaz local siendo su próximo destino el *hook ingress*.
- 6) El paquete llega al *hook ingress* asociado a la interfaz “*lo*”, se le asigna un nuevo ID de *nftrace* (741c1ec7) y se contabiliza, siendo su veredicto *continue*.
- 7) El paquete continúa su camino por el *hook prerouting*, donde hace *match* con la primera regla, aumentando el contador y generando el *log* “Pasa por pre-inet”, su veredicto es *continue*.
- 8) Se toma la decisión de *routing* del paquete y se determina que este, es para la interfaz “*lo*”.
- 9) El paquete llega al *hook input*, donde hace *match* con la primera regla, generando el *log* “Pasa por input-inet”, siendo su veredicto *accept*.

5. [Equipo:Encaminador - Usuario:dit] Se comprueba el monitor de nft para ver el transcurso del paquete reply.

```
trace id 34be81ac inet [filter output] rule ip protocol icmp nftrace set 1 (verdict continue)
trace id 34be81ac inet [filter output] rule ip protocol icmp log prefix "Pasa por output-inet: " counter packets 21 bytes 1764 accept (verdict accept)
trace id 34be81ac inet filter postrouting packet: oif "lo" ip saddr 127.0.0.1 ip daddr 127.0.0.1 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 27600 ip length 84
mp type echo-reply icmp code 0 icmp id 4578 icmp sequence 1
trace id 34be81ac inet filter postrouting rule ip protocol icmp log prefix "Pasa por post-inet: " counter packets 27 bytes 2268 accept (verdict accept)
trace id 344401a0 netdev filteringress in-lo packet: iif "lo"
trace id 344401a0 netdev filteringress in-lo rule ip protocol icmp counter packets 2 bytes 168 (verdict continue)
trace id 344401a0 netdev filteringress in-lo verdict continue
trace id 344401a0 netdev filteringress in-lo
trace id 344401a0 netdev filteringress in-lo
trace id 344401a0 [filter prerouting] packet: iif "lo" ip saddr 127.0.0.1 ip daddr 127.0.0.1 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 27600 ip length 84 ip
p type echo-reply icmp code 0 icmp id 4578 icmp sequence 1
trace id 344401a0 inet filter prerouting rule ip protocol icmp log prefix "Pasa por pre-inet: " counter packets 27 bytes 2268 accept (verdict accept)
trace id 344401a0 inet [filter input] packet: iif "lo" ip saddr 127.0.0.1 ip daddr 127.0.0.1 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 27600 ip length 84 icmp t
echo-reply icmp code 0 icmp id 4578 icmp sequence 1
trace id 344401a0 inet filter input rule ip protocol icmp log prefix "Pasa por input-inet: " counter packets 21 bytes 1764 accept (verdict accept)
```

Figura 4.17 - Nftrace ICMP rep interfaz lo

Camino del ICMP reply (ver Figura 4.17):

- 1) El paquete ICMP rep, es generado por el proceso local como respuesta al request recibido.
- 2) El paquete llega al *hook output*, donde se comienza a hacer el seguimiento del mismo, se le asigna el identificador (34be81ac).
- 3) Tras activar la traza de *nftrace* del paquete, se le aplica la segunda regla, aumentando el contador de esta y generando un *log* con el nombre “Pasa por output-inet”, su veredicto es *continue*.
- 4) El paquete llega al *hook postrouting*, donde hace *match* con la primera regla y genera un *log* con el nombre “Pasa por post-inet”, su veredicto es *continue*.
- 5) El paquete sale por la interfaz local siendo su próximo destino el *hook ingress*.
- 6) El paquete llega al *hook ingress* asociado a la interfaz “*lo*”, se le asigna un nuevo ID de *nftrace* (344401a0) y se contabiliza, siendo su veredicto *continue*.

- 7) El paquete continúa su camino por el *hook prerouting*, donde hace *match* con la primera regla, aumentando el contador y generando el *log* “Pasa por pre-inet”, su veredicto es *continue*.
 - 8) Se toma la decisión de *routing* del paquete y se determina, que este es para la interfaz “lo”.
 - 9) El paquete llega al *hook input*, donde hace *match* con la primera regla, generando el *log* “Pasa por input-inet”, siendo su veredicto *accept*.
6. **[Equipo:Encaminador - Usuario:dit]** Se comprueba el *ruleset* de nftables, para ver el incremento de los contadores de la cadena *ingress* de la interfaz “lo” y se comprueba la captura de Wireshark.

En las *Figura 4.18* y *Figura 4.19* se pueden ver respectivamente, como se ha incrementado el contador de la cadena *ingress* asociada a la interfaz “lo” y la captura de Wireshark donde se puede comprobar el ID del paquete ICMP.

```
table netdev filteringress {
    chain ingress {
        type filter hook ingress device ens38 priority 0; policy accept;
        ip protocol icmp counter packets 0 bytes 0
    }

    chain in-lo {
        type filter hook ingress device lo priority 0; policy accept;
        ip protocol icmp counter packets 2 bytes 168
    }
}
```

Figura 4.18 - Regla ingress interfaz lo

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) request id=0x11e2, seq=1/256, ttl=64 (reply in 2)
2	0.000102595	127.0.0.1	127.0.0.1	ICMP	98	Echo (ping) reply id=0x11e2, seq=1/256, ttl=64 (request in 1)

► Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
 ► Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
 ► Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
 ▾ Internet Control Message Protocol
 Type: 0 (Echo (ping) reply)
 Code: 0
 Checksum: 0x7cae [correct]
 [Checksum Status: Good]
 Identifier (BE): 4578 (0x11e2)
 Identifier (LE): 57873 (0xe211)
 Sequence number (BE): 1 (0x0001)
 Sequence number (LE): 256 (0x0100)
 [Request frame: 1]
 [Response time: 0,103 ms]
 Timestamp from icmp data: Jan 20, 2021 01:27:57.000000000 CET
 [Timestamp from icmp data (relative): 0.181213318 seconds]
 ► Data (48 bytes)

Figura 4.19 - Captura Wireshark interfaz lo

7. **[Equipo:Encaminador - Usuario:root]** Se procede a comprobar, que los *logs* generados por el transcurso del paquete, se corresponden con los indicados en el punto 5.

Equipo:Encaminador Usuario:root

```
$ cat /var/log/kern.log | grep Pasa
```

```

localhost kernel: [48833.243642] device lo entered promiscuous mode
localhost kernel: [48926.194809] Pasa por output-inet: IN= OUT=lo SRC=127.0.0.1 DST=127.0.0.1 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=27599 DF PROTO=ICMP TYPE=8 CODE=0 ID=4578 SEQ=1
localhost kernel: [48926.194826] Pasa por post-inet: IN= OUT=lo SRC=127.0.0.1 DST=127.0.0.1 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=27599 DF PROTO=ICMP TYPE=8 CODE=0 ID=4578 SEQ=1
localhost kernel: [48926.194901] Pasa por pre-inet: IN=lo OUT= MAC=00:00:00:00:00:00 SRC=127.0.0.1 DST=127.0.0.1 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=27599 DF PROTO=ICMP TYPE=8 CODE=0 ID=4578 SEQ=1
localhost kernel: [48926.194924] Pasa por input-inet: IN=lo OUT= MAC=00:00:00:00:00:00 SRC=127.0.0.1 DST=127.0.0.1 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=27599 DF PROTO=ICMP TYPE=8 CODE=0 ID=4578 SEQ=1
localhost kernel: [48926.194955] Pasa por output-inet: IN= OUT=lo SRC=127.0.0.1 DST=127.0.0.1 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=27600 DF PROTO=ICMP TYPE=0 CODE=0 ID=4578 SEQ=1
localhost kernel: [48926.194964] Pasa por post-inet: IN= OUT=lo SRC=127.0.0.1 DST=127.0.0.1 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=27600 DF PROTO=ICMP TYPE=0 CODE=0 ID=4578 SEQ=1
localhost kernel: [48926.194996] Pasa por pre-inet: IN=lo OUT= MAC=00:00:00:00:00:00 SRC=127.0.0.1 DST=127.0.0.1 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=27600 DF PROTO=ICMP TYPE=0 CODE=0 ID=4578 SEQ=1
localhost kernel: [48926.242151] Pasa por input-inet: IN=lo OUT= MAC=00:00:00:00:00:00 SRC=127.0.0.1 DST=127.0.0.1 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=27600 DF PROTO=ICMP TYPE=0 CODE=0 ID=4578 SEQ=1

```

Figura 4.20 - Registro logs ICMP interfaz lo

En la Figura 4.20, se observa el registro de *logs* dejado por el paquete ICMP y a través del cual, se pueden confirmar los puntos por los que han atravesado, hasta realizar el ping por la interfaz “*lo*”.

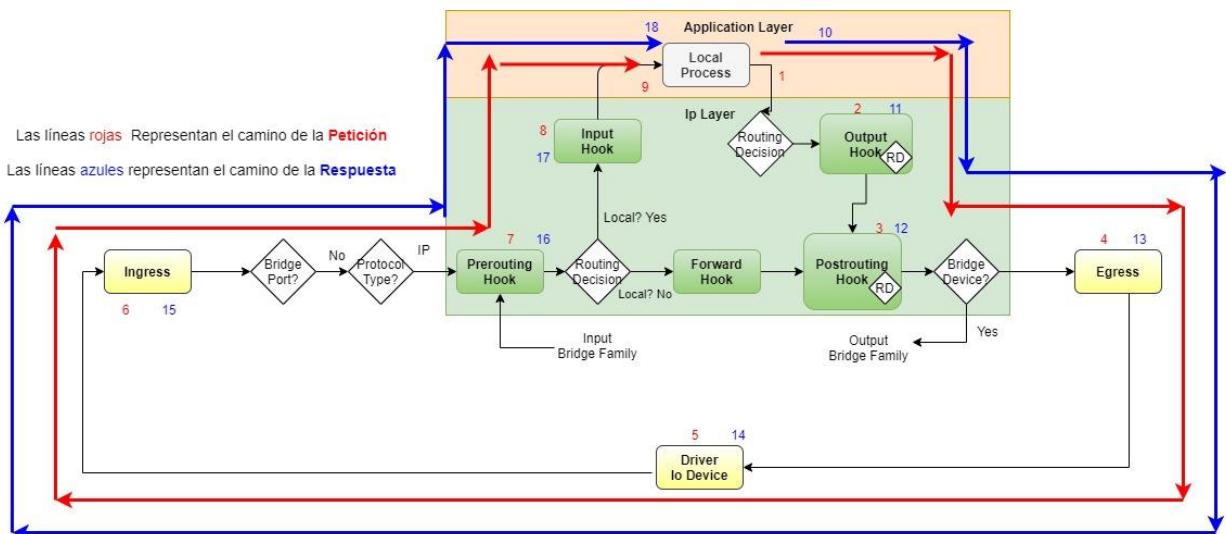


Figura 4.21 - Resumen camino ping lo

En la Figura 4.21, se puede observar paso a paso el camino que ha seguido el paquete, desde su generación por el kernel hasta llegar a este.

Los pasos se encuentran marcados con números, e indican el orden en el cual se pasan por los *hooks*, del mismo modo, el color rojo representa la petición, mientras que el color azul se usa para la respuesta.

Conclusión del ejemplo:

Como se ha podido comprobar, el hecho de realizar el ping hacia la dirección de la interfaz “lo”, implica que tanto el paquete de pregunta como el de respuesta, realizan dos vueltas completas por los *hooks* de Netfilter (salvando el *hook forward*, que implicaría el reenvío).

Este ejemplo, es muy ilustrativo, ya que si vemos ejemplos en la red, del camino de estos paquetes, nos encontramos con multitud de ejemplo imprecisos, que ofrecen una visión sesgada o errónea del camino que sigue el paquete “lo”.

5 MANAGERS

En este capítulo, se analizará que es y cómo funciona, un manager de reglas con nftables.

Los managers, son *frontend* que ayudan a administrar y configurar las reglas del *firewall*, en este caso de nftables.

Estos tienen la capacidad de enmascarar la complejidad de la creación, distribución y mantenimiento de las reglas del *framework* de Netfilter.

Actualmente los managers de reglas más conocidos, han dado su respuesta acerca de si van a migrar a esta nueva herramienta o por el contrario solo seguirán dando soporte a iptables.

Nos encontramos en un escenario en el cual, la mayoría de las herramientas para manejo de reglas se encuentra actualmente migrando o con vistas de migrar en un futuro.

Managers tan conocidos como; Ipfire[28], Shorewall o Ferm[29], han optado por no migrar y solo ofrecen su herramienta con soporte para iptables.

La respuesta de estas, están basadas en que para ofrecer soporte a nftables, no basta con parchear la aplicación existente, sino que el cambio de sintaxis de la herramienta, exige la generación de una nueva herramienta para adaptarse a ella.

Por otro lado, existen managers que han decidido migrar, pero todavía se encuentran en una fase temprana del proyecto, incluyendo solo parte de las funcionalidades que estos ofrecían con iptables, es el caso de Suricata y Firewallld, los cuales ya ofrecen la posibilidad de trabajar con nftables.

Existe otro manager muy conocido como es Fail2ban, el cual ofrece soporte para nftables de manera nativa, pero está teniendo bastantes complicaciones en su desarrollo, provocando que los usuarios reporten una considerable cantidad de fallos en su Github.

Con todo lo mencionado, e independientemente del estado de actualización en el que se encuentren, todas adolecen del mismo problema y es la falta de nueva documentación, que explique los cambios y la nueva forma de emplear estos managers.

Por lo que para este punto, se ha decidido mostrar el uso de un manager que, aunque actualmente no tiene la funcionalidad al 100% con nftables, sí que se considera que cubre con solvencia áreas que antes se hacían con iptables, ese manager es Firewallld.

5.1 Firewalld

Como se ha comentado anteriormente, Firewalld es un manager que nos permite gestionar las reglas de nftables.

En este apartado, se explica su funcionamiento con ejemplos y se verán las reglas que este va creando en nftables.

La instalación de esta herramienta es algo complejo debido a sus dependencias, por ello se explica en su totalidad en el “Anexo D - Instalación y Configuración Firewalld”.

Lo que cabe destacar de este manager, es que la gestión de reglas las hace mediante el uso, de lo que él define como “zonas”[30].

Estas zonas, no son más que agrupaciones de reglas que dependen del nivel de confianza que se le otorgue al tráfico de red.

Firewalld trabaja con 9 zonas de confianza, las cuales van desde la menos confiable “*drop*” hasta la de mayor confianza, conocida como “*trusted*”

La Tabla 7, recoge esas zonas y una breve descripción de cada una de ellas.

Zona	Descripción
Drop	Es la zona de menor confianza y en ella se realiza el <i>drop</i> de todos los paquetes que lleguen al equipo.
Block	Esta zona es parecida a Drop, con la diferencia de que en lugar de realizar el <i>drop</i> de los paquetes, lo que hace es un <i>reject</i> , informando al emisor con un mensaje ICMP.
Public	Es la más alta de las zonas de no confianza. En ella indicamos que no confiamos en los paquetes que recibimos, salvo que exista una regla específica que los permita.
External	Este tipo de zona se utiliza para realizar la traducción de direcciones del tráfico. De forma que nuestro equipo se comportará como frontera entre lo que considera zona privada y zona pública. En esta zona, al igual que en Public, desconfiamos del tráfico, salvo que existan reglas específicas.
Internal	Es la otra cara de la zona External, en ella entendemos que nos encontramos en nuestra zona privada y generalmente confiaremos en los paquetes que nos lleguen, estando ya algunos servicios permitidos de forma básica desde esta zona.
DMZ	Se usa para configurar zonas desmilitarizadas y al igual que Public, necesitamos reglas específicas para aceptar o denegar tráfico.
Work	Se usa para identificar un área de trabajo en la que confiamos, de forma que la mayoría de servicios se encuentran permitidos de forma básica en esta zona.
Home	La zona Home, es aquella en la que tenemos casi la totalidad de servicios disponibles para su acceso y solo unos pocos requieren reglas específicas para poder acceder.
Trusted	Es la Zona de mayor confianza y en él se encuentra permitido todo el tráfico.

Tabla 7 - Zonas Firewalld

5.1.1 Escenario de Pruebas

Ejemplo prueba manager.

Para ejemplificar el uso de esta herramienta, se usará el escenario de dos equipos.

La creación y configuración de este escenario se indica en el “Anexo E - Escenario 2 Equipos”.

Descripción del ejemplo:

Como ejemplo, se configurará en el equipo Encaminador ambas tarjetas en zona Internal, que como se ha visto, usa una política restrictiva.

En ella se habilitará el ping, los protocolos DNS[31], HTTP[32] y HTTPS y habilitaremos el puerto 80 para poder recibir peticiones por este.

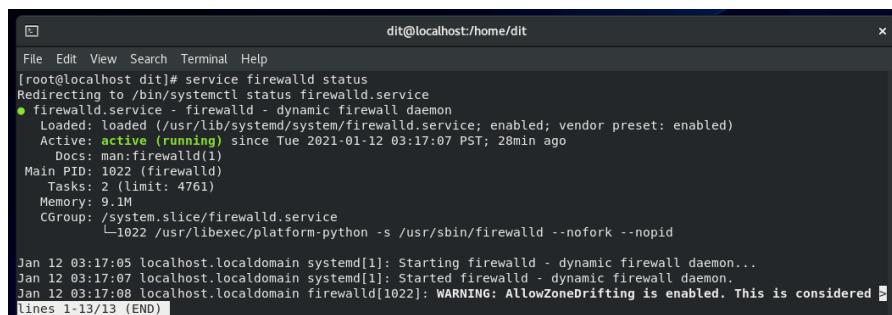
Para ello se activa el servicio httpd para que el servidor Apache[33] que se encuentra en el equipo Encaminador arranque y comience a escuchar peticiones TCP en el puerto 80.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Lo primero que se debe comprobar es el estado de servicio de Firewallld, para ello se usa el siguiente comando:

```
Equipo:Encaminador Usuario:root
```

```
$ service firewalld status
```



A terminal window titled 'dit@localhost:/home/dit' showing the output of the 'service firewalld status' command. The output indicates that the firewalld service is active and running. It provides details such as the main PID (1022), tasks (2), memory usage (9.1M), and the cgroup path (/system.slice/firewalld.service). Log entries at the bottom show the service starting and a warning about AllowZoneDrifting being enabled.

```
dit@localhost:/home/dit
File Edit View Search Terminal Help
[root@localhost dit]# service firewalld status
Redirecting to /bin/systemctl status firewalld.service
● firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2021-01-12 03:17:07 PST; 28min ago
       Docs: man:firewalld(1)
   Main PID: 1022 (firewalld)
      Tasks: 2 (limit: 4761)
     Memory: 9.1M
      CGroup: /system.slice/firewalld.service
              └─1022 /usr/libexec/platform-python -s /usr/sbin/firewalld --nofork --nrepid

Jan 12 03:17:05 localhost.localdomain systemd[1]: Starting firewalld - dynamic firewall daemon...
Jan 12 03:17:07 localhost.localdomain systemd[1]: Started firewalld - dynamic firewall daemon.
Jan 12 03:17:08 localhost.localdomain firewalld[1022]: WARNING: AllowZoneDrifting is enabled. This is considered ...
(lines 1-13/13 (END))
```

Figura 5.1 - Estado servicio Firewallld

En la Figura 5.1, se puede comprobar como el servicio se encuentra en estado *running*.

Una vez comprobado que este se encuentra activo, se puede proceder con el resto de pasos.

2. **[Equipo:Encaminador - Usuario:root]** Los siguientes comandos, permitirán saber la zona en la que se encuentra nuestro equipo, así como las zonas que se encuentran activas.

```
Equipo:Encaminador Usuario:root
```

```
$ firewall-cmd --get-default-zone
$ firewall-cmd --get-active-zones
$ firewall-cmd --zone=public --list-all
```

```

dit@localhost:~$ firewall-cmd --get-default-zone
public
dit@localhost:~$ firewall-cmd --set-default-zone=internal
success
dit@localhost:~$ firewall-cmd --get-active-zones
internal
    interfaces: ens33 ens37
libvirt
    interfaces: virbr0
dit@localhost:~$ firewall-cmd --zone=public --list-all
public
    target: default
    icmp-block-inversion: no
    interfaces:
    sources:
    services: cockpit dhcpcv6-client http ssh
    ports:
    protocols:
    masquerade: no
    forward-ports:
    source-ports:
    icmp-blocks:
    rich rules:
dit@localhost:~$
```

Figura 5.2 - Zonas definidas básicas

Se puede observar el resultado de los comandos anteriormente ejecutados en la *Figura 5.2*.

- 1) “--get-default-zone”, indica que zona es la que se encuentra activada por defecto.
- 2) “--get-active-zones” muestra las zonas que se encuentran activas y que por lo tanto, serán las que cursen el tráfico.

En este caso, vemos como las zonas activas son : *internal* y *libvirt*, siendo la segunda de ellas una zona especial que se usa para interfaces virtuales.

Este muestra también a que zona se encuentran asociadas nuestras NIC’s.

Esto es así, porque por defecto, tenemos que asociar cada interfaz a una zona de confianza, cuyo equivalente en nftables sería, darle una política por defecto.

- 3) “--zone=public --list-all” podemos observar las propiedades de la zona public, la cual muestra; las interfaces asociadas, puertos permitidos, protocolos etc...
3. [Equipo:Encaminador - Usuario:root] Se procede a asociar las interfaces que van a ser usadas para que pertenezcan a la zona Internal.

Equipo:Encaminador Usuario:root

```
$ firewall-cmd --zone=internal --change-interface=ens37
$ firewall-cmd --zone=internal --change-interface=ens33
```

4. [Equipo:Encaminador - Usuario:root] A continuación, se habilita en la zona Internal el servicio HTTP y HTTPS con su puerto habilitado para la escucha.

Equipo:Encaminador Usuario:root

```
$ firewall-cmd --zone=internal --add-service=http --permanent
$ firewall-cmd --zone=internal --add-service=https --permanent
$ firewall-cmd --zone=internal --add-port=80/tcp --permanent
$ firewall-cmd --zone=internal --add-port=80/udp --permanent
```

5. [Equipo:Encaminador - Usuario:root] Por último, se añaden servicios básicos de comunicación; Samba, DNS y SSH.

```
Equipo:Encaminador Usuario:root
```

```
$ firewall-cmd --permanent --zone=internal --add-service=samba  
$ firewall-cmd --permanent --zone=internal --add-service=dns  
$ firewall-cmd --permanent --zone=internal --add-service=ssh
```

6. [Equipo:Encaminador - Usuario:root] Una vez realizado esto, se procede a listar la configuración de la zona Internal con el siguiente comando.

```
Equipo:Encaminador Usuario:root
```

```
$ firewall-cmd --zone=internal --list-all
```

```
[dit@localhost ~]$ firewall-cmd --zone=internal --list-all  
internal (active)  
  target: default  
  icmp-block-inversion: no  
  interfaces: ens33 ens37  
  sources:  
  services: cockpit dhcpcv6-client http https mdns samba-client ssh  
  ports: 80/tcp  
  protocols:  
  masquerade: no  
  forward-ports:  
  source-ports:  
  icmp-blocks:  
  rich rules:
```

Figura 5.3 - Configuración zona Internal

Como se puede apreciar en la Figura 5.3, la configuración recoge todas las opciones que se le han indicado mediante los comandos anteriores.

7. [Equipo:Encaminador - Usuario:root] Se procede a arrancar el servidor Apache de la maquina Encaminador.

```
Equipo:Encaminador Usuario:root
```

```
$ service httpd start  
$ service httpd status
```

```
[dit@localhost ~]$ service httpd start  
Redirecting to /bin/systemctl start httpd.service  
[dit@localhost ~]$ service httpd status  
Redirecting to /bin/systemctl status httpd.service  
● httpd.service - The Apache HTTP Server  
  Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor pres  
    Active: active (running) since Wed 2021-01-13 02:40:08 PST; 2s ago  
      Docs: man:httpd.service(8)  
    Main PID: 3510 (httpd)  
      Status: "Started, listening on: port 80"  
        Tasks: 213 (limit: 4761)  
      Memory: 24.7M  
     CGroup: /system.slice/httpd.service  
           ├─3510 /usr/sbin/httpd -DFOREGROUND  
           ├─3521 /usr/sbin/httpd -DFOREGROUND  
           ├─3524 /usr/sbin/httpd -DFOREGROUND  
           ├─3526 /usr/sbin/httpd -DFOREGROUND  
           ├─3527 /usr/sbin/httpd -DFOREGROUND
```

Figura 5.4 - Arranque del servidor Apache

En la Figura 5.4, se comprueba que el servidor se encuentra levantado y activo.

A continuación, se comprobará la conectividad, conectándose desde PC1 al servidor Apache del Encaminador.

8. [Equipo:PC1 - Usuario:root] Desde PC1, se intenta la conexión a la URL: <http://10.10.1.20:80>

Equipo: PC1 Usuario:root

```
$ curl 10.10.1.20
```

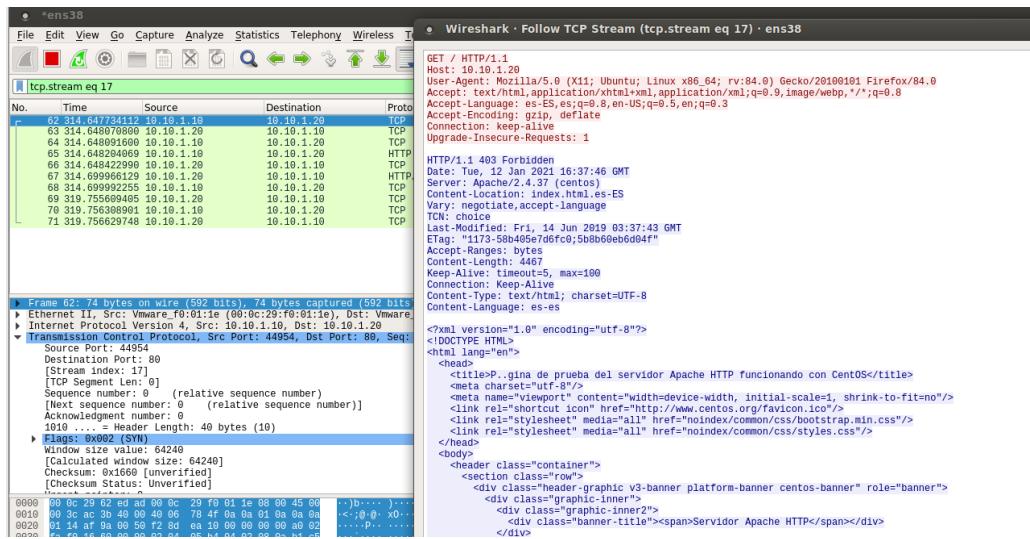


Figura 5.5 - Flujo conexión Apache (Internal)

Como se puede ver en las Figura 5.5 y Figura 5.6, a PC1 llegan los paquetes de conexión que sirven la página web de Apache, pudiendo comprobar que esta se muestra en el buscador.



Test Page

This page is used to test the proper operation of the [Apache HTTP server](#) after it has been installed. If you can read this page it means that this site is working properly. This server is powered by [CentOS](#).

Just visiting?

The website you just visited is either experiencing problems or is undergoing routine maintenance.

If you would like to let the administrators of this website know that you've seen this page instead of the page you expected, you should send them e-mail. In general, mail sent to the name "webmaster" and directed to the website's domain should reach the appropriate person.

For example, if you experienced problems while visiting www.example.com, you should send e-mail to "webmaster@example.com".

Important note:

The CentOS Project has nothing to do with this

Are you the Administrator?

You should add your website content to the directory </var/www/html/>.

To prevent this page from ever being used, follow the instructions in the file </etc/httpd/conf.d/welcome.conf>.

Promoting Apache and CentOS

You are free to use the images below on Apache and CentOS Linux powered HTTP servers. Thanks for using Apache and CentOS!



Figura 5.6 - Página test Apache(Internal)

9. [Equipo:Encaminador - Usuario:root] Una vez realizado este ejemplo, se probará a realizar una redirección de puertos, para ello, se añade la siguiente regla a la zona Internal.

```
Equipo:Encaminador Usuario:root
```

```
$firewall-cmd --zone="public" --add-forward port=port=100: \proto=tcp:toport=80
```

Con esta regla, se habilita que se pueda llegar al Servidor Apache desde peticiones que se realicen al puerto 100 de la máquina Encaminador.

10. [Equipo:PC1 - Usuario:dit] Se realiza una nueva petición desde PC1, esta vez con la siguiente Url: <http://10.10.1.20:100> y se recarga la página del buscador.

```
Equipo:PC1 Usuario:root
```

```
$ hping3 -S 10.10.1.20 -p 100
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.10.1.10	10.10.1.20	TCP	74	46968 - 100 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSeqval=2981799671 TSeqr=0 WS=128
2	0.000320166	10.10.1.10	10.10.1.10	TCP	74	100 - 46968 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSeqval=2981799671 TSeqr=13422 TSeqr=29
3	0.000336358	10.10.1.10	10.10.1.20	TCP	66	46968 - 100 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSeqval=2981799672 TSeqr=13422
4	0.000442202	10.10.1.10	10.10.1.20	HTTP	446	GET / HTTP/1.1
5	0.000617370	10.10.1.20	10.10.1.10	TCP	66	100 - 46968 [ACK] Seq=1 Ack=381 Win=30080 Len=0 TSeqval=13423 TSeqr=2981799672
6	0.00424897	10.10.1.20	10.10.1.10	HTTP/X...	4963	HTTP/1.1 403 Forbidden
7	0.004249015	10.10.1.10	10.10.1.20	TCP	66	46968 - 100 [ACK] Seq=381 Ack=4899 Win=61568 Len=0 TSeqval=2981799676 TSeqr=13426
8	0.040667495	10.10.1.10	10.10.1.20	HTTP	512	GET /noindex/common/images/pb-apache.png HTTP/1.1
9	0.042466278	10.10.1.10	10.10.1.20	TCP	74	46970 - 100 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSeqval=2981799714 TSeqr=0 WS=128
10	0.042570649	10.10.1.20	10.10.1.10	HTTP	249	HTTP/1.1 304 Not Modified
11	0.042674983	10.10.1.20	10.10.1.10	TCP	74	100 - 46970 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSeqval=13465 TSeqr=29
12	0.042687449	10.10.1.10	10.10.1.20	TCP	66	46970 - 100 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSeqval=2981799714 TSeqr=13465
13	0.042788910	10.10.1.10	10.10.1.20	HTTP	511	GET /noindex/common/images/pb-centos.png HTTP/1.1
14	0.043778585	10.10.1.20	10.10.1.10	HTTP	247	HTTP/1.1 304 Not Modified
15	0.086552259	10.10.1.10	10.10.1.20	TCP	66	46968 - 100 [ACK] Seq=1272 Ack=5261 Win=64128 Len=0 TSeqval=2981799758 TSeqr=13466

Frame 15: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
Ethernet II, Src: VMware_0:01:1e (00:0c:29:f0:01:1e), Dst: VMware_62:ed:ad (00:0c:29:62:ed:ad)
Internet Protocol Version 4, Src: 10.10.1.10, Dst: 10.10.1.20
Transmission Control Protocol, Src Port: 46968, Dst Port: 100, Seq: 1272, Ack: 5261, Len: 0
Source Port: 46968
Destination Port: 100
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 1272 (relative sequence number)
[Next sequence number: 1272 (relative sequence number)]
Acknowledgment number: 5261 (relative ack number)
1000 = Header Length: 32 bytes (8)
Flags: 0x010 (ACK)
Window size value: 501
[Calculated window size: 64128]
[Window size scaling factor: 128]
Checksum: 0x1658 [unverified]

Figura 5.7 - Redirección de puertos (Internal)

Como se puede ver en la Figura 5.7, el equipo PC1 lanza una petición a Encaminador con puerto destino 100, respondiéndonos y sirviendo la web de Apache.

La aplicación de estas reglas y zonas de confianza, se hace posible gracias a que la herramienta trabaja de manera interna con la estructura de nftables.

Para comprobar las reglas que Firewalld ha creado en nftables, basta con abrir un terminal, pasar a modo super usuario e indicar el siguiente comando.

11. [Equipo:Encaminador - Usuario:dit] Se comprueban las reglas creadas en nftables

```
Equipo:Encaminador Usuario:dit
```

```
$ sudo su
$ nft list ruleset
```

```

chain filter_IN_libvirt_allow {
    udp dport 67 ct state { new, untracked } accept
    udp dport 547 ct state { new, untracked } accept
    tcp dport 53 ct state { new, untracked } accept
    udp dport 53 ct state { new, untracked } accept
    tcp dport 22 ct state { new, untracked } accept
    udp dport 69 ct helper set "helper-tftp-udp"
    udp dport 69 ct state { new, untracked } accept
    meta l4proto icmp ct state { new, untracked } accept
    meta l4proto ipv6-icmp ct state { new, untracked } accept
}

```

Figura 5.8 - Reglas libvirt Firewallld

En la Figura 5.8 se puede observar las reglas que Firewallld crea por defecto con respecto a interfaces virtuales.

Estas interfaces se configuran de forma básica en la zona de confianza Home, por lo que permite acceso a la mayoría de servicios desplegados, estos servicios son:

Puerto	Servicio
67 UDP	DHCP modo sin conexión
547 UDP	DHCPv6
53 TCP/UDP	DNS
22 TCP	SSH
69 TCP/UDP	TFTP

Tabla 8 - Puertos y servicios libvirt Firewallld

Como se puede ver en la Tabla 8, existe un servicio que tiene un tratamiento distinto, TFTP, esto se debe a que este protocolo opera con el puerto de control 69 TCP/UDP, pero el puerto de transferencia de datos es elegido por el usuario.

Por ello sería complicado poder realizar el seguimiento de la conexión, cuando esta se produce en dos puertos con dos canales de comunicación distintos.

Para evitar este tipo de problemas, se añade un *helper* de connection-tracking, el cual permitirá asociar el tráfico de los dos puertos como si fuesen un único flujo de tráfico.

Este tipo de configuración se verá con detalle en el punto “6.7 Helpers”.

El siguiente bloque de reglas que Firewallld crea, son las reglas de la zona de confianza con las que se ha elegido trabajar (ver Figura 5.9).

```

chain filter_IN_internal_allow {
    tcp dport 22 ct state { new, untracked } accept
    ip daddr 224.0.0.251 udp dport 5353 ct state { new, untracked } accept
    ip6 daddr ff02::fb udp dport 5353 ct state { new, untracked } accept
    udp dport 137 ct helper set "helper-netbios-ns-udp"
    udp dport 137 ct state { new, untracked } accept
    udp dport 138 ct state { new, untracked } accept
    ip6 daddr fe80::/64 udp dport 546 ct state { new, untracked } accept
    tcp dport 9090 ct state { new, untracked } accept
    tcp dport 80 ct state { new, untracked } accept
    tcp dport 443 ct state { new, untracked } accept
}

```

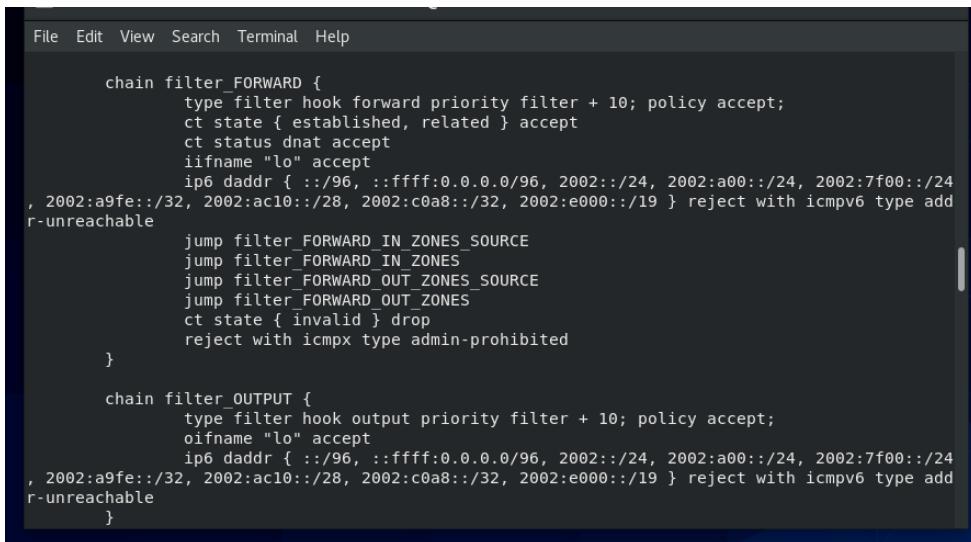
Figura 5.9 - Reglas zona Internal Firewallld

Las reglas creadas en esta cadena se corresponden con la cadena input de la zona de confianza Internal.

En ella se puede ver como se ha permitido el tráfico de; SSH, DHCP, Webmsg (9090), HTTP, TLS (443) y la entrada de comunicación de NetBIOS, la cual al igual que TFTP, usa dos canales (uno para control y otro para datos), es por este motivo, por el que se usa el helper.

De esta manera permite realizar el seguimiento de la conexión, pese a usar dos puertos diferentes (2 flujos paralelos).

Por último, se comprobará en la Figura 5.10, la creación de las reglas por defecto de *forwarding* y *output*.



```

File Edit View Search Terminal Help

chain filter_FORWARD {
    type filter hook forward priority filter + 10; policy accept;
    ct state { established, related } accept
    ct status dnat accept
    iifname "lo" accept
    ip6 daddr { ::/96, ::ffff:0.0.0.0/96, 2002::/24, 2002:a00::/24, 2002:7f00::/24
, 2002:a9fe::/32, 2002:ac10::/28, 2002:c0a8::/32, 2002:e000::/19 } reject with icmpv6 type add
r-unreachable
    jump filter_FORWARD_IN_ZONES_SOURCE
    jump filter_FORWARD_IN_ZONES
    jump filter_FORWARD_OUT_ZONES_SOURCE
    jump filter_FORWARD_OUT_ZONES
    ct state { invalid } drop
    reject with icmpx type admin-prohibited
}

chain filter_OUTPUT {
    type filter hook output priority filter + 10; policy accept;
    oifname "lo" accept
    ip6 daddr { ::/96, ::ffff:0.0.0.0/96, 2002::/24, 2002:a00::/24, 2002:7f00::/24
, 2002:a9fe::/32, 2002:ac10::/28, 2002:c0a8::/32, 2002:e000::/19 } reject with icmpv6 type add
r-unreachable
}

```

Figura 5.10 - Reglas forward Firewallld

Al igual que en los casos anteriores, realiza el seguimiento de todas las conexiones permitidas o denegadas.

Además, deniega una serie de direcciones Ipv6 que se consideran de entorno privado, o que son paquetes malformados que esperan recabar información de nuestro equipo.

Este es el funcionamiento que tiene Firewallld para crear reglas nftables dependiendo de la zona de confianza asignada.

También se ha podido comprobar que podemos aplicar reglas más específicas para aceptar o denegar tráfico.

El archivo de configuración con todas las reglas creadas, así como la distribución de las cadenas y la tabla de Firewallld, están en el “Anexo D - Instalación y Configuración Firewallld”.

Este manager también permite introducir reglas desde su propia línea de comandos.

Pero esta función solo se encuentra disponible por el momento para iptables, siendo esta su principal línea de desarrollo para un futuro.

La adaptación del manager como hemos podido comprobar es casi completa, pero le sigue faltando la potencia que ofrecía con iptables, pudiendo introducir reglas específicas usando la sintaxis de iptables.

Firewalld tiene previsto que esta funcionalidad, salga en versión de pruebas con la actualización de junio de 2021, pudiendo encontrarse en versión estable para finales de este año.

Conclusión del ejemplo:

Este ejemplo sirve para ilustrar la configuración y uso de un manager.

Como se ha podido ver, estos todavía no cuentan con las mismas funcionalidades que las ofrecidas para su uso con iptables (En este caso, no permite la inclusión de reglas por líneas de comandos para el manager).

Como se comentó en la introducción de esta guía, estos proyectos se encuentran en un nivel de madurez, poco avanzada.

Muchos de estos proyectos, han decidido no adaptarse a esta nueva herramienta y los que están adaptándose, se encuentran todavía en una fase beta.

La conclusión que se puede sacar es que existen proyectos muy interesantes, que en un futuro podrán ofrecer una funcionalidad muy completa y competitiva con iptables, pero que actualmente no pueden competir, debido a la poca madurez de los mismos o a los numerosos problemas que estos presentan (*bugs*).

6 SEGUIMIENTO DE CONEXIONES(Conntrack)

En este capítulo, se analizará uno de los módulos más importantes de Netfilter, el módulo Conntrack[35].

Este módulo, se encarga de realizar el seguimiento del flujo de conexiones en nuestro equipo.

Conntrack, es el encargado de hacer que nuestra herramienta se comporte como un *Stateful firewall* [36], que son aquellos *firewalls*, que no solo se encargan de inspeccionar la cabecera de los paquetes, sino que también, pueden discernir entre flujos de comunicación.

Este concepto es muy útil a la hora de evitar ciertos tipos de ataques, como pueden ser ataques de escaneo o de denegación de servicios.

Antes de comenzar a ver los puntos que conforman este capítulo, se recomienda revisar los siguientes anexos:

- Anexo H - Repaso Protocolos (TCP,UDP,ICMP)
- Anexo I - Concepto Conexión

Con esos anexos se repasarán los conceptos básicos de los protocolos que se van a ver, así como clarificar y explicar el concepto de conexiones en conntrack.

6.1 Máquina de estados.

La máquina de estados[37], es un concepto que usa Netfilter, para poder clasificar e identificar el estado de cada paquete que llegue a nuestro equipo.

Como se ha comentado al principio de este capítulo, es esta capacidad, la que permite que Netfilter trabaje como un *Stateful-firewall*.

Estos estados pueden tomar 5 valores distintos, los cuales se detallan en la Tabla 9:

Estado	Descripción
NEW	Es el estado inicial de una conexión. Se asigna a un paquete, si este pertenece a una secuencia válida de iniciación de la conexión o si es, el primer paquete válido que ve el <i>firewall</i> de un flujo de comunicación (ejemplo: paquete SYN en una conexión TCP). En este estado el <i>firewall</i> solo ha visto tráfico en una dirección de la comunicación.

ESTABLISHED	Este estado se otorga cuando la conexión ha sido establecida, es decir, el <i>firewall</i> , ha visto tráfico en ambos sentidos de la comunicación.
RELATED	<p>Se considera este estado como un estado de expectación.</p> <p>Este se usa para decir que un paquete de un flujo de comunicación, está relacionado con otro ya existente.</p> <p>Cumple la misma función que NEW, pero en flujos de comunicación que dependen de un flujo maestro, por ejemplo , el flujo de comunicación de datos FTP, el cual depende del flujo de comunicación de control FTP.</p>
INVALID	Este estado se asigna a paquetes, que no cumplen el estado de expectación o que no están relacionados con ningún flujo de comunicación, bien porque no puedan realizar la apertura del mismo, o porque no pertenezcan a ningún flujo existente.
UNTRACKED	<p>Este estado se les asigna a aquellos paquetes que, mediante una regla, se le aplica la opción <i>notrack</i>.</p> <p>De esta forma, no se realizará el seguimiento de esos paquetes, ni se dejará constancia de su flujo de comunicación en una entrada de conntrack.</p>

Tabla 9 - Estados conntrack

Estos estados vienen referidos, dentro de conntrack, con la opción *state*.

Una vez conocido los estados en los que puede encontrarse los paquetes, de una conexión determinada, se deben aclarar ciertos puntos.

El hecho de que conntrack realice el seguimiento de conexiones, no significa que únicamente sirva para detectar y establecer flujos de comunicación TCP, el cual, si es un protocolo orientado a conexión, sino que además también lo usan protocolos como UDP e ICMP.

Esto se debe a que, aunque el módulo se refiera a seguimiento de conexiones, lo que en verdad realiza es el seguimiento de flujos de comunicación.

A continuación, se detallarán los estados vistos en la Tabla 9.

- **NEW:** Como se ha comentado anteriormente, este es el primer estado de una conexión y el cuál se le otorga al paquete que la inicia.
A pesar, de que en muchos lugares se puede leer que este estado se le adjudica al primer paquete que se ve de una conexión, esto no es del todo cierto, ya que, para poder crear una tupla de conexión, este paquete debe ser de inicio de conexión en el caso de TCP, o de tipo request en el caso de ICMP.
Si se encuentran ante un flujo UDP, cualquier paquete perteneciente a este tipo de flujos que no haya sido registrado con anterioridad, obtendrá el estado de NEW.
- **ESTABLISHED:** Este estado se otorga a aquellos paquetes, que pertenecen a un flujo de comunicación ya establecido, es decir, que ese flujo ya ha sido registrado por un paquete con el estado NEW.

- **RELATED:** Este estado puede ser el más complejo de entender, ya que es el usado para las comunicaciones *multi-flow*, aquellas que usan varios flujos de comunicación.

Un ejemplo de este tipo de comunicaciones es el protocolo FTP [38], el cual, usa un canal de control y otro canal distinto de datos.

Este tipo de estado se les otorga a aquellos paquetes que se encuentren relacionados con un flujo de comunicación existente, pero que tienen su propio flujo de comunicación.

Un paquete con este estado, actúa como un paquete con el estado NEW, es decir, sirve para iniciar un flujo de comunicación, la diferencia radica en que este flujo de comunicación, depende o se encuentra relacionado con otro ya existente.

- **INVALID:** Este estado se les otorga a aquellos paquetes , que no se pueden adjudicar a ningún flujo de conexiones existentes, o bien, el tipo de estos no les permite crear una nueva entrada en la tabla de conexiones.

Al no poder realizar ninguna de las dos acciones, internamente se catalogan como inválidos.

- **UNTRACKED:** Este estado se les otorga aquellos paquetes, que por motivos de seguridad o por razones específicas del entorno, no se desea que creen entrada dentro de conntrack.

Para hacer que un paquete no cree entrada en conntrack, es decir, que no se busque identificar su flujo de comunicación, se usa la opción *notrack* [39] de nftables.

Una vez visto los diferentes estados que puede tomar un paquete, es necesario indicar en que puntos del flow de paquetes se realiza esta clasificación de estados.

Esta clasificación de estados, tiene lugar en dos puntos concretos, en el *hook prerouting* en la prioridad -200 y en el *hook output* en la prioridad -200.

Esto quiere decir que la clasificación tiene lugar dentro de estos *hooks*, en una prioridad muy baja, lo que le da preferencia ante reglas de cadenas como; filter, NAT, mangle etc...

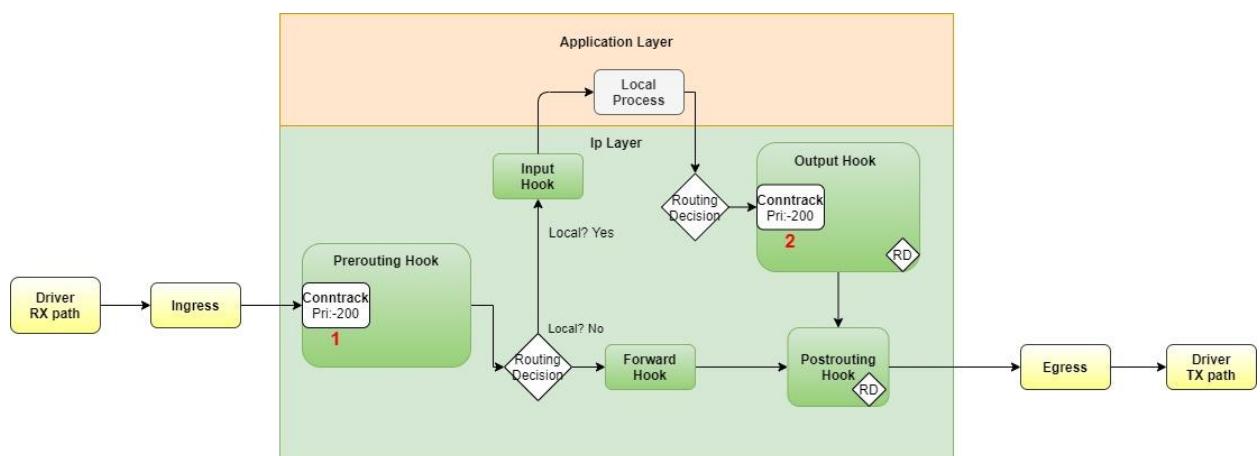


Figura 6.1 - Puntos de clasificación de los paquetes con conntrack

En la Figura 6.1, se pueden ver los puntos en los cuales trabaja conntrack, para asignar los diferentes estados a los paquetes.

La asignación de estados tiene lugar dentro de los *hooks* indicados en la prioridad -200, si se añaden reglas, que usen el módulo conntrack y que tengan una prioridad más baja que la -200 en el *hook de prerouting* o en el *hook output*, se podrá ver como estas reglas nunca hacen *match*, al no haber realizado todavía la clasificación del paquete.

6.1.1 Ejemplo de clasificación de estados

Descripción del ejemplo.

En este ejemplo, se busca probar el punto de clasificación de estados de conntrack, para ello, se crearán dos cadenas output, una de ellas con prioridad 0 y la otra con una prioridad de -250 y se añadirá una regla para contabilizar el número de paquetes con estado NEW que pasen por estas.

Para este ejemplo, no hará falta usar ningún escenario, basta con usar el equipo Encaminador para realizar la prueba.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Se crea una tabla filter y dos cadenas asociadas al *hook output* dentro de ella, una con prioridad 0, y otra con prioridad -250.

```
Equipo:Encaminador Usuario:root

$ nft add table ip filter
$ nft add chain ip filter 'output { type filter hook output priority
0 ; }'
$ nft add chain ip filter 'out { type filter hook output priority
-250 ; }'
```

2. **[Equipo:Encaminador - Usuario:root]** Se añaden dos reglas, una a cada cadena creada, en la cual si el protocolo es ICMP y el estado de este es NEW aumente su contador.

```
Equipo:Encaminador Usuario:root

$ nft add rule ip filter output ip protocol icmp ct state new
counter accept
$ nft add rule ip filter out ip protocol icmp ct state new counter
accept
```

3. **[Equipo:Encaminador - Usuario:root]** Se muestra la *ruleset* de nftables y se comprueba que su salida sea igual a la mostrada en la Figura 6.2.

```
Equipo:Encaminador Usuario:root

$ nft list ruleset
```

```

root@localhost:/home/dit# nft list ruleset
table ip filter {
    chain out {
        type filter hook output priority -250; policy accept;
        ip protocol icmp ct state new counter packets 0 bytes 0 accept
    }

    chain output {
        type filter hook output priority 0; policy accept;
        ip protocol icmp ct state new counter packets 0 bytes 0 accept
    }
}
root@localhost:/home/dit#

```

Figura 6.2 - Estructura de cadenas y reglas del ejemplo de clasificación

4. [Equipo:Encaminador - Usuario:root] Se realiza un ping a la dirección de la máquina pública 8.8.8.8.

```
Equipo:Encaminador Usuario:root
```

```
$ ping -c 1 8.8.8.8
```

5. [Equipo:Encaminador - Usuario:root] Se muestra la *ruleset* de nftables y se comprueba, que contador ha aumentado y el por qué de este aumento.

```
Equipo:Encaminador Usuario:root
```

```
$ nft list ruleset
```

```

root@localhost:/home/dit# nft list ruleset
table ip filter {
    chain out {
        type filter hook output priority -250; policy accept;
        ip protocol icmp ct state new counter packets 0 bytes 0 accept
    }

    chain output {
        type filter hook output priority 0; policy accept;
        ip protocol icmp ct state new counter packets 1 bytes 84 accept
    }
}
root@localhost:/home/dit#

```

Figura 6.3 - Contadores de las reglas del ejemplo de clasificación de estados

Como se puede ver en la Figura 6.3, solo ha aumentado el contador de la regla cuya cadena tiene una prioridad mayor que -200 (punto en el cual se clasifican los estados de conntrack).

La regla que se encuentra en la cadena con la prioridad -250 todavía no ha sido capaz de clasificar el estado del paquete .

Conclusión del ejemplo:

Este ejemplo ha servido para ilustrar el punto exacto, en el cual, Netfilter, realiza la categorización del paquete (Figura 6.1).

Sabiendo, que esta se produce en la prioridad -200, si un administrador de redes, desea trabajar con reglas de filtrado, según el estado de los paquetes, deberá crear cadenas, cuya prioridad sea mayor a la -200, en caso contrario y tal como se ha podido comprobar, los paquetes no realizarán *match* con estas reglas, independientemente del estado que se busque comprobar.

Antes de pasar a ver cómo trabaja cada protocolo con el sistema de seguimiento de conexiones, se explicará, como funcionan las tuplas de conntrack, así como los datos que esta toma de cada protocolo para realizar el seguimiento de conexiones.

6.2 Connection tracking Linux

Conntrack se encarga de adquirir y almacenar los datos necesarios de los paquetes, así como almacenarlos en una estructura que le permitirá discernir sus flujos de comunicación.

De forma general, los datos que el módulo conntrack toma de nuestros paquetes son los siguientes:

- Dirección origen y destino
- Puerto origen y destino
- Tipo de protocolo

Si el protocolo es ICMP, los datos que almacena conntrack varían con respecto a los que se acaban de mencionar.

Tomando conntrack los siguientes datos:

- Dirección origen y destino
- Tipo y Código del ICMP
- ID del ICMP
- Tipo de protocolo

Además de estos datos, en la estructura de conexiones que mantiene conntrack, se observan otros campos que aportan información adicional:

- El status de la conexión
- El *timeout* de la tupla creada
- La marca del flujo de comunicación
- El flag de asegurado (ASSURED)
- El flag de “sin respuesta”, el cual nos indica que solo se ha visto comunicación en una dirección (UNREPLIED)
- El número de flujos que hacen referencia a esa entrada (use)

Una vez vistos los datos básicos, que conntrack toma de nuestros paquetes, se procederá a ver qué es y como se crean las tuplas de los diferentes flujos de conexión [40].

Se han explicado los datos y la máquina de estados de Netfilter, falta indicar como se organiza la tabla de conexiones y su funcionamiento en conjunto con el kernel.

Para poder trabajar con la tabla de conexiones, Netfilter se basa en 4 funciones, estas funciones son:

- La creación del flujo de conexión
- La inspección de los paquetes de un flujo
- La manipulación de los paquetes fragmentados

- El uso de *helpers* [41].

La tabla de conexiones, se divide en particiones llamadas “buckets”[42] y cada una de estas particiones está formada por una lista doblemente enlazada, en la que cada miembro es una tupla de tipo *hash*.

Para cada conexión, se tienen 2 tuplas distintas.

- Una tupla que representa los datos en la dirección originaria de la conexión (aquella que la inicia)
- Una segunda tupla con los datos de la respuesta.

Cada una de las tuplas, contiene la información relevante de la conexión.

Se debe entender, que la conexión a la que se refiere el sistema, es distinta al concepto de conexión de un protocolo como puede ser TCP.

Mientras que TCP es un protocolo que se encuentra orientado a conexiones, lo que connection tracking conoce como “*connection*”, no es más que la representación de un flujo de datos con origen y destino, el cual puede usar un protocolo orientado o no, a conexión (TCP o UDP entre otros).

Esta explicación viene detallada en el “Anexo I - Concepto Conexión” .

Como se ha comentado anteriormente, la unidad más importante de este módulo son las tuplas, las cuales se usan para representar flujos unidireccionales de comunicación.

Esta estructura contiene las uniones, en las cuales se guardan los datos que resultan de interés para realizar el seguimiento de dicho flujo.

Las diferentes uniones, son las estructuras en las cuales se almacenan los valores de las cabeceras de los siguientes protocolos: TCP, UDP, ICMP, DCCP[43] y SCTP[44].

Estas estructuras, independientes para cada protocolo, guardan la información que se considera única para identificar un paquete, es por ello, que, si se realiza el seguimiento de un paquete TCP o UDP, guardará los puertos origen y destino, mientras que en caso de realizar el seguimiento de paquetes ICMP, guardará el tipo, el código, así como también su identificador.

Además de estos valores, conntrack necesita también conocer información específica de los protocolos, de capa 3 y capa 4 que tenga el paquete.

Es de esta manera, por la cual el módulo de conntrack, toma los datos de los paquetes que atraviesan los *hooks* de Netfilter, para después organizarlos en estructuras dentro de una tupla.

Para evitar la complejidad de mantener todos los datos de cada paquete guardado en cada una de las tuplas del sistema, conntrack usa lo que se denomina como tupla *hash*, la cual consiste en realizar el *hash* de los valores de las estructuras, guardando en la tupla este valor *hash*.

Las dos tuplas *hash* (una para cada dirección de conexión) se guardan en una estructura que se denomina como “*nf_conn*”[45].

Las entradas creadas o actualizadas con las que trabaja el módulo conntrack, se pueden imprimir por pantalla usando la herramienta conntrack-tool.

Una vez visto el funcionamiento de conntrack y como este, guarda la información y crea las tuplas, se procederá a ver en qué puntos crea estas entradas.

Como se ha visto en el punto “Máquina de estados.”, los paquetes se clasifican dentro del *hook prerouting* y del *hook output*, cabría preguntarse si las entradas de conntrack son creadas en estos mismos puntos.

La respuesta es no.

Las entradas en la tabla de conntrack se crean en el *hook input* y en el *hook postrouting*, con un valor de prioridad máximo.

Si recordamos como funciona la prioridad, “Prioridades de las Cadenas”, el valor máximo representa la última prioridad a ejecutar, por lo que esta, es la última operación que tiene lugar en el *hooks input* y en el *hook postrouting*.

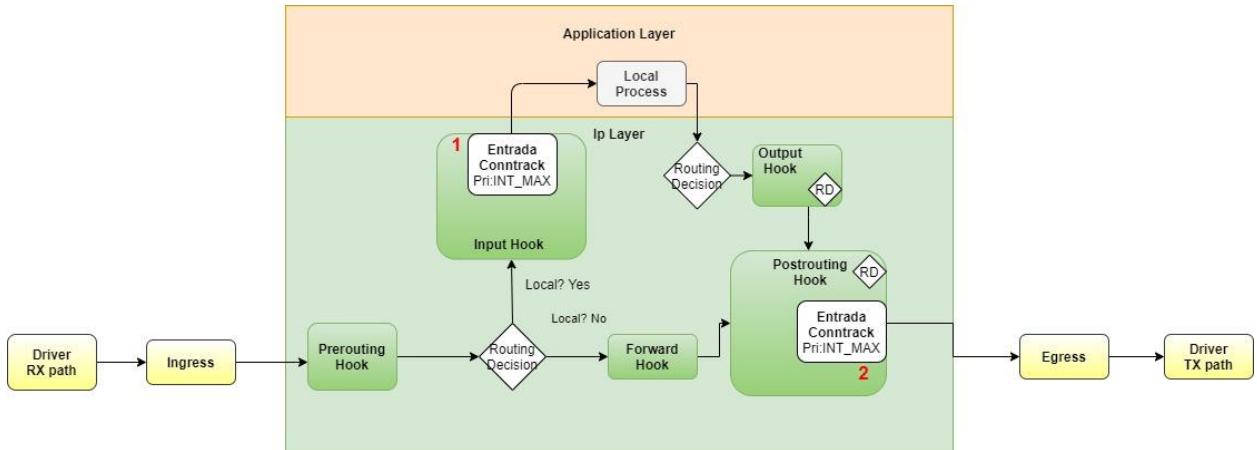


Figura 6.4 - Creación de las entradas conntrack

En la Figura 6.4, se puede observar los puntos en los cuales se crean las entradas de conntrack, estos se encuentran marcados con un número, se procede a explicar el significado de las mismas:

- 1) Si tenemos un paquete, cuyo destino es un proceso local de la máquina, este entrar por la NIC del equipo, será procesado por el *hook ingress* y el *hook prerouting*, se tomará la decisión de encaminamiento del mismo y al ser para la máquina local, este se procesará por el *hook input*. Dentro de este *hook*, la entrada se creará en el valor de prioridad máximo, es decir, en la última acción que se realice dentro del *hook input* y antes de pasar el paquete al proceso local.
- 2) Si tenemos un paquete cuyo destino no es un proceso local, sino que este, ha de ser encaminado, tras la decisión de *routing*, viajará por el *hook forward* y el *hook postrouting*, siendo en este último y con valor de prioridad máxima donde se creará la entrada. Este punto también sirve para crear o actualizar las entradas de los paquetes generados por el proceso local.

Conntrack-tools, es la encargada de imprimirnos la información que antes podía comprobarse en la ruta “/proc/net/ip_conntrack”[46], la cual se encontraba muy limitada, ya que solo permitía imprimir las entradas existentes de la tabla conntrack.

Conntrack-tool y su herramienta conntrack, permiten ahora , no solo imprimir esa información, sino manejarla(actualizarla, crear una entrada, borrar una entrada etc...).

En caso de que se desee ver las entradas del módulo conntrack, se debe usar el siguiente comando:

Equipo: Encaminador **Usuario:** root

```
$ conntrack -L
```

Conntrack, también ofrece una opción que permite imprimir la tabla, con la misma salida, que la impresa por “/proc/net/ip_conntrack”.

Equipo: Encaminador **Usuario:** root

```
$ conntrack -L -o extended
```

Una vez visto el comando, se pasará a indicar los diferentes campos que conforman una entrada TCP, UDP e ICMP.

6.2.1 Entrada conntrack TCP

```
root@localhost:/home/dit# conntrack -L -o extended
ipv4      2 tcp      6 111 TIME_WAIT src=10.10.1.20 dst=10.10.1.10 sport=54662 dport=80 src=10.10.1.10 dst=10.10.1.20 sport=80 dport=54662 [ASSURED] mark=2 use=1
conntrack v1.4.6 (conntrack-tools); 1 flow entries have been shown.
```

Figura 6.5 - Entrada conntrack TCP extendido

Analizamos la información mostrada en la Figura 6.5 (Entrada TCP):

- 1) Para comenzar, se tiene el protocolo de capa 3, que en este caso es el ipv4
- 2) Representación de este protocolo de forma numérica decimal, el valor en este caso es “2”.
- 3) Se tiene la información del protocolo de capa 4, el cual se puede ver que es TCP
- 4) Representación del número de este protocolo en notación decimal “6”.
- 5) Después viene el tiempo de vida restante para dicha entrada: en la Figura 6.5 se puede observar que a esta entrada de conexión le quedan 111 segundos, los cuales, irán disminuyendo hasta que se vea más tráfico perteneciente a dicha conexión.

Cuando llegue nuevo tráfico, el contador de esta entrada, se establece de nuevo a su valor por defecto.

- 6) El siguiente dato, es la información del estado actual en que se encuentra la entrada. En este caso se puede comprobar que la tupla se encuentra en el estado TIME_WAIT. El valor interno de una conexión es ligeramente distinto a los empleados externamente por nftables.

El valor TIME_WAIT indica que se encuentra esperando tráfico para esa entrada en concreto, este tipo de estados depende del protocolo que se esté usando y se verán en el punto específico de cada protocolo.

- 7) Dirección IP de origen del paquete de petición.
- 8) Dirección IP de destino del paquete de petición.
- 9) Puerto de origen del paquete de petición.
- 10) Puerto de destino del paquete de petición.
- 11) Pasados estos 4 campos, encontramos el valor de un flag, en la Figura 6.5, el valor de esta es 0. Este flag indica que solo se ha visto tráfico en una dirección y se denomina UNREPLIED.

- 12) Dirección IP de origen de la respuesta.
- 13) Dirección IP de destino de la respuesta.
- 14) Puerto de origen de la respuesta.
- 15) Puerto de destino de la respuesta.

- 16) En caso de haber visto tráfico en ambas direcciones, se borra el valor del flag UNREPLIED y se sustituye por uno que se encuentra en el antepenúltimo valor de la entrada, “ASSURED”, el cual indica que se ha visto tráfico en ambos sentidos.

Este flag nos indica que la entrada se encuentra asegurada y que ante un llenado de todas las entradas de conntrack esta no se borrará, al contrario de lo que ocurriría con la entrada cuyo flag es “UNREPLIED”.

- 17) Mark, nos da la marca de ese flujo de comunicación.
- 18) Use, nos indica el número de usos de ese entrada de flujo.

Como se puede comprobar en la Figura 6.5, para crear los datos de la entrada pertenecientes a la respuesta,

conntrack toma las direcciones de origen y destino y los invierte con respecto a los vistos en el comienzo de la entrada, lo mismo ocurre con los puertos de origen y destino que también se encuentran invertidos respecto a los anteriores.

Tabla de parámetros de una entrada conntrack TCP		
1º	<i>Protocolo de capa 3</i>	ipv4
2º	<i>Numeración protocolo capa 3 (Decimal)</i>	2
3º	<i>Protocolo de capa 4</i>	TCP
4º	<i>Numeración protocolo capa 4 (Decimal)</i>	6
5º	<i>Timeout de la entrada (Duración)</i>	111
6º	<i>State interno de la entrada</i>	TIME_WAIT
7º	<i>IP origen (original)</i>	10.10.1.20
8º	<i>IP destino (original)</i>	10.10.1.10
9º	<i>Puerto origen (original)</i>	54662
10º	<i>Puerto destino (original)</i>	80
11º	<i>Flag UNREPLIED</i>	0
12º	<i>IP origen (reply)</i>	10.10.1.10
13º	<i>IP destino (reply)</i>	10.10.1.20
14º	<i>Puerto origen (reply)</i>	80
15º	<i>Puerto destino (reply)</i>	54662
16º	<i>Flag ASSURED</i>	1
17º	<i>Marca de la conexión</i>	2
18º	<i>Use de la entrada</i>	1

Tabla 10 - Campos de la entrada conntrack TCP

En la Tabla 10, se pueden observar los diferentes valores que tienen los campos de la entrada TCP de conntrack.

6.2.2 Entrada conntrack UDP

```
root@localhost:/home/dit# conntrack -L -o extended
ipv4      2 udp      17 24 src=192.168.106.137 dst=8.8.8.8 sport=52565 dport=53 src=8.8.8.8 dst=192.168.106.137 sport=53 dport=52565 mark=0 use=1
```

Figura 6.6 - Entrada conntrack UDP extendido

Analizamos la información mostrada en la Figura 6.6 (Entrada UDP):

- 1) Para comenzar, se tiene el protocolo de capa 3, que en este caso es el ipv4
- 2) Representación de este protocolo de forma numérica decimal, el valor en este caso es “2”.
- 3) Se tiene la información del protocolo de capa 4, el cual se puede ver que es UDP
- 4) Representación del número de este protocolo en notación decimal “17”.
- 5) Después viene el tiempo de vida restante para dicha entrada, en la Figura 6.6 se puede observar que a esta entrada de conexión le quedan 24 segundos, los cuales, irán disminuyendo hasta que se vea más tráfico perteneciente a dicha conexión.

Cuando llegue nuevo tráfico, el contador de esta entrada se establece de nuevo a su valor por defecto.

- 6) Dirección IP de origen del paquete de petición.
- 7) Dirección IP de destino del paquete de petición.
- 8) Puerto de origen del paquete de petición.
- 9) Puerto de destino del paquete de petición.
- 10) Pasados estos 4 campos, podemos encontrarnos el valor de un flag , en la Figura 6.6 el valor de esta es 0.

Este flag indica que solo se ha visto tráfico en una dirección y se denomina UNREPLIED.

- 11) Dirección IP de origen de la respuesta.
- 12) Dirección IP de destino de la respuesta.
- 13) Puerto de origen de la respuesta.
- 14) Puerto de destino de la respuesta.
- 15) Mark, nos da la marca de ese flujo de comunicación.
- 16) Use, nos indica el número de usos de ese entrada de flujo.

Tabla de parámetros de una entrada conntrack UDP		
1º	<i>Protocolo de capa 3</i>	ipv4
2º	<i>Numeración protocolo capa 3 (Decimal)</i>	2
3º	<i>Protocolo de capa 4</i>	UDP
4º	<i>Numeración protocolo capa 4 (Decimal)</i>	17
5º	<i>Timeout de la entrada</i>	24

	<i>(Duración)</i>	
6º	<i>IP origen (original)</i>	192.168.106.137
7º	<i>IP destino (original)</i>	8.8.8.8
8º	<i>Puerto origen (original)</i>	52565
9º	<i>Puerto destino (original)</i>	53
10º	<i>Flag UNREPLIED</i>	0
11º	<i>IP origen (reply)</i>	8.8.8.8
12º	<i>IP destino (reply)</i>	192.168.106.137
13º	<i>Puerto origen (reply)</i>	53
14º	<i>Puerto destino (reply)</i>	52565
15º	<i>Marca de la conexión</i>	0
16º	<i>Use de la entrada</i>	1

Tabla 11 - Campos de la entrada conntrack UDP

En la Tabla 11, se pueden observar los diferentes valores que tienen los campos de la entrada UDP de conntrack.

6.2.3 Entrada conntrack ICMP

```
root@localhost:/home/dit# conntrack -L -o extended
ipv4    2 icmp   1 28 src=192.168.106.137 dst=216.58.215.131 type=8 code=0 id=2854 src=216.58.215.131 dst=192.168.106.137 type=0 code=0 id=2854 mark=0 use=1
```

Figura 6.7 - Entrada conntrack ICMP extendido

Analizamos la información mostrada en la Figura 6.7 (Entrada ICMP):

- 1) Para comenzar, se tiene el protocolo de capa 3, que en este caso es el ipv4 .
- 2) Representación de este protocolo de forma numérica decimal, el valor en este caso es “2”.
- 3) Se tiene la información del protocolo de capa 4, el cual se puede ver que es ICMP
- 4) Representación del número de este protocolo en notación decimal “1”.
- 5) Después viene el tiempo de vida restante para dicha entrada, en la Figura 6.7 se puede observar que a esta entrada de conexión le quedan 28 segundos, los cuales, irán disminuyendo hasta que se vea más tráfico perteneciente a dicha conexión.

Cuando llegue nuevo tráfico, el contador de esta entrada se establece de nuevo a su valor por defecto.

- 6) Dirección IP de origen del paquete de petición.
- 7) Dirección IP de destino del paquete de petición.
- 8) Tipo ICMP del paquete de petición.

9) Código ICMP del paquete de petición.

10) ID ICMP del paquete de petición

Pasados estos 5 campos, podemos encontrarnos el valor de un flag , en la Figura 6.7 el valor de esta es 0.

Este flag indica que solo se ha visto tráfico en una dirección y se denomina UNREPLIED.

11) Dirección IP de origen de la respuesta.

12) Dirección IP de destino de la respuesta.

13) Tipo ICMP del paquete de respuesta.

14) Código ICMP del paquete de respuesta.

15) ID ICMP del paquete de respuesta.

16) Mark, nos da la marca de ese flujo de comunicación.

17) Use, nos indica el número de usos de ese entrada de flujo.

17) Tabla de parámetros de una entrada conntrack		
1º	<i>Protocolo de capa 3</i>	ipv4
2º	<i>Numeración protocolo capa 3 (Decimal)</i>	2
3º	<i>Protocolo de capa 4</i>	ICMP
4º	<i>Numeración protocolo capa 4 (Decimal)</i>	1
5º	<i>Timeout de la entrada (Duración)</i>	28
6º	<i>IP origen (original)</i>	192.168.106.137
7º	<i>IP destino (original)</i>	216.58.215.131
8º	<i>Tipo ICMP (original)</i>	8
9º	<i>Código ICMP (original)</i>	0
10º	<i>ID ICMP (original)</i>	2854
11º	<i>Flag UNREPLIED</i>	0
12º	<i>IP origen (reply)</i>	216.58.215.131
13º	<i>IP destino (reply)</i>	192.168.106.137
14º	<i>Tipo ICMP (reply)</i>	0
15º	<i>Código ICMP (reply)</i>	0
16º	<i>ID ICMP (reply)</i>	2854

17º	<i>Marca de la conexión</i>	0
18º	<i>Módulos que usan la entrada</i>	1

Tabla 12 - Campos de la entrada conntrack ICMP

En la Tabla 12, se pueden observar los diferentes valores que tienen los campos de la entrada ICMP de conntrack.

Una vez se ha visto los distintos tipos de entradas que podemos tener y los campos que las conforman, pasaremos a ver el uso de conntrack en modo monitor, con la opción -E.

Equipo: Encaminador Usuario: root
\$ conntrack -E

En caso de usarlo así, debemos indicar que existen unos estados específicos que veremos a continuación.

```
[NEW] udp      17 30 src=192.168.106.128 dst=192.168.106.254 sport=68 dport=67 [UNREPLIED] src=192.168.106.128 dst=192.168.106.254 sport=68 dport=67
[UPDATE] udp      17 30 src=192.168.106.128 dst=192.168.106.254 sport=68 dport=67 src=192.168.106.254 dst=192.168.106.128 sport=68 dport=67
[DESTROY] tcp      6 src=10.10.1.20 dst=193.147.162.130 sport=42826 dport=80 src=193.147.162.130 dst=193.147.162.130 sport=80 dport=42826
```

Figura 6.8 - Estados entradas conntrack modo monitor

En la Figura 6.8, se pueden ver los 3 valores de estados que aparecen al comienzo de la entrada; *NEW*, *UPDATE* y *DESTROY*, estos estados se encuentran descritos en la Tabla 13.

Estos valores a diferencia de los vistos en la Tabla 9, no representan el estado de un paquete, sino que representan el estado de una entrada dentro de conntrack.

Estado	Descripción
NEW	<p>Este estado nos indica que la entrada acaba de crearse dentro de conntrack.</p> <p>Por lo que, al crearse esta entrada, entendemos que el paquete que la ha creado, ha atravesado el <i>hook input</i> si su destino es la máquina local o el <i>hook postrouting</i> si este ha sido encaminado por el equipo.</p> <p>Estos paquetes son los primeros pertenecientes a un nuevo flujo de comunicación que ve conntrack.</p> <p>Esta entrada se caracteriza porque siempre tendrá activado el flag UNREPLIED.</p>
UPDATE	<p>Este estado nos indica que se ha actualizado una entrada ya existente en conntrack.</p> <p>La actualización, se da cada vez que se ve un paquete perteneciente a un flujo de comunicación preexistente.</p>
DESTROY	<p>Este estado, se da cuando expira el temporizador que tiene la entrada de conntrack y nos indica, que esa entrada se ha borrado del sistema.</p>

Tabla 13 - Estados entradas conntrack

6.2.4 Ejemplo de Seguimiento de Conexiones

Descripción del ejemplo:

En este ejemplo, se busca mostrar el uso básico de la herramienta conntrack, de forma que se pueda monitorizar los eventos de las entradas de esta, así como ver y buscar por un determinado patrón en las entradas que tenga guardadas.

Para realizar este ejemplo, se usará el escenario de dos equipos “Anexo E - Escenario 2 Equipos”.

La instalación de conntrack-tools [47] [48], se especifica en el “Anexo C - Instalación Conntrack-Tools”.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Se comprueba que la tabla de conntrack se encuentra vacía con el siguiente comando y se observa si la salida coincide con la mostrada en la Figura 6.9.

```
Equipo: Encaminador Usuario: root  
$ conntrack -L
```

```
root@localhost:/home/dit# conntrack -L  
conntrack v1.4.6 (conntrack-tools): 0 flow entries have been shown.
```

Figura 6.9 - Mostrar entradas conntrack

Como se puede observar en la Figura 6.9, se encuentra inicialmente vacía, para que nuestro módulo comience a guardar las conexiones, es imprescindible que el tráfico sea procesado por algún *hook* de nftables.

2. **[Equipo:Encaminador - Usuario:root]** Para comenzar a ver las entradas de conntrack, basta con añadir una regla que contabilice el tráfico y se vuelve a comprobar la tabla de conexiones.

```
Equipo: Encaminador Usuario: root  
  
$ nft add table ip filter  
$ nft add chain ip filter prerouting { type filter hook prerouting priority 0 ; }  
$ nft add rule ip filter prerouting counter
```

3. **[Equipo:Encaminador - Usuario:root]** Para que PC1 pueda comunicarse con máquinas públicas a través de Encaminador, se añaden las siguientes reglas.

```
Equipo: Encaminador Usuario: root  
  
$ nft add table ip nat  
$ nft add chain ip nat prerouting { type nat hook prerouting priority 0 ; }  
$ nft add chain ip nat postrouting { type nat hook postrouting priority 100 ; }  
$ nft add rule ip nat prerouting ip daddr 192.168.106.128 dnat to 10.10.1.20  
$ nft add rule ip nat postrouting ip saddr 10.10.1.0/24 counter masquerade
```

En las Figura 6.10 y Figura 6.11, se puede ver como queda la estructura de cadenas y reglas de este ejemplo.

```
root@localhost:/home/dit# nft add table ip filter
root@localhost:/home/dit# nft add chain' ip filter prerouting { type filter hook prerouting priority 0 ; }'
root@localhost:/home/dit# nft add rule ip filter prerouting counter
root@localhost:/home/dit# nft list ruleset
table ip filter {
    chain prerouting {
        type filter hook prerouting priority 0; policy accept;
        counter packets 0 bytes 0
    }
}
```

Figura 6.10 - Añadir regla counter en prerouting

```
root@localhost:/home/dit# nft list ruleset
table ip nat {
    chain postrouting {
        type nat hook postrouting priority 100; policy accept;
        ip saddr 10.10.1.0/24 counter packets 1663 bytes 108238 masquerade
    }
    chain prerouting {
        type nat hook prerouting priority 0; policy accept;
        ip daddr 192.168.106.128 dnat to 10.10.1.20
    }
}
table ip filter {
    chain prerouting {
        type filter hook prerouting priority 0; policy accept;
        counter packets 284 bytes 38359
    }
}
root@localhost:/home/dit#
```

Figura 6.11 - Reglas NAT escenario 2 equipos

4. **[Equipo:PC1 - Usuario:dit]** Una vez añadidas las reglas, nos conectamos a firefox en PC1 y navegamos para generar tráfico de red.
5. **[Equipo:Encaminador - Usuario:root]** Una vez generado el tráfico de red, volvemos a Encaminador y mostramos las entradas de conntrack con el siguiente comando.

Equipo:Encaminador Usuario:root

```
$ conntrack -L
```

En la Figura 6.12, podemos observar, como ahora si podemos visualizar las entradas de la tabla de conexiones, las cuales se han ido añadiendo, conforme se ha navegado por la red.

```
tcp   6 151955 ESTABLISHED src=10.10.1.20 dst=91.198.174.208 sport=48192 dport=443 src=91.198.174.208 dst=192.168.106.128 sport=443 dport=48192 [ASSURED] mark=0 use=1
conntrack v1.4.6 (conntrack-tools): 20 flow entries have been shown.
root@localhost:/home/dit# conntrack -L
tcp   6 5 CLOSE src=10.10.1.20 dst=91.198.174.208 sport=48192 dport=443 src=91.198.174.208 dst=192.168.106.128 sport=443 dport=48192 [ASSURED] mark=0 use=1
tcp   17 116 src=10.10.1.20 dst=8.8.8.8 sport=54961 dport=53 src=8.8.8.8 dst=192.168.106.128 sport=53 dport=54961 [ASSURED] mark=0 use=1
tcp   17 177 src=10.10.1.20 dst=8.8.8.8 sport=51346 dport=53 src=8.8.8.8 dst=192.168.106.128 sport=53 dport=51346 [ASSURED] mark=0 use=1
tcp   6 431992 ESTABLISHED src=10.10.1.20 dst=34.223.172.12 sport=55352 dport=443 src=34.223.172.12 dst=192.168.106.128 sport=443 dport=55352 [ASSURED] mark=0 use=1
tcp   6 118 TIME_WAIT src=10.10.1.20 dst=91.198.174.192 sport=39592 dport=443 src=91.198.174.192 dst=192.168.106.128 sport=443 dport=39592 [ASSURED] mark=0 use=1
tcp   17 141 src=10.10.1.20 dst=8.8.8.8 sport=42767 dport=53 src=8.8.8.8 dst=192.168.106.128 sport=53 dport=42767 [ASSURED] mark=0 use=1
tcp   17 116 src=10.10.1.20 dst=8.8.8.8 sport=35188 dport=53 src=8.8.8.8 dst=192.168.106.128 sport=53 dport=35188 [ASSURED] mark=0 use=1
tcp   17 116 src=10.10.1.20 dst=8.8.8.8 sport=46954 dport=53 src=8.8.8.8 dst=192.168.106.128 sport=53 dport=46954 [ASSURED] mark=0 use=1
tcp   17 139 src=10.10.1.20 dst=8.8.8.8 sport=44704 dport=53 src=8.8.8.8 dst=192.168.106.128 sport=53 dport=44704 [ASSURED] mark=0 use=1
tcp   6 115 TIME_WAIT src=10.10.1.20 dst=91.198.174.208 sport=48194 dport=443 src=91.198.174.208 dst=192.168.106.128 sport=443 dport=48194 [ASSURED] mark=0 use=1
tcp   6 52 TIME_WAIT src=10.10.1.20 dst=13.224.119.71 sport=59504 dport=443 src=13.224.119.71 dst=192.168.106.128 sport=443 dport=59504 [ASSURED] mark=0 use=1
tcp   6 431965 ESTABLISHED src=10.10.1.20 dst=151.101.1.44 sport=33094 dport=443 src=151.101.1.44 dst=192.168.106.128 sport=443 dport=33094 [ASSURED] mark=0 use=1
tcp   6 5 CLOSE src=10.10.1.20 dst=91.198.174.208 sport=48196 dport=443 src=91.198.174.208 dst=192.168.106.128 sport=443 dport=48196 [ASSURED] mark=0 use=1
tcp   17 116 src=10.10.1.20 dst=8.8.8.8 sport=50516 dport=53 src=8.8.8.8 dst=192.168.106.128 sport=53 dport=50516 [ASSURED] mark=0 use=1
tcp   17 112 src=10.10.1.20 dst=8.8.8.8 sport=38926 dport=53 src=8.8.8.8 dst=192.168.106.128 sport=53 dport=38926 [ASSURED] mark=0 use=1
tcp   6 115 TIME_WAIT src=10.10.1.20 dst=91.198.174.208 sport=48198 dport=443 src=91.198.174.208 dst=192.168.106.128 sport=443 dport=48198 [ASSURED] mark=0 use=1
```

Figura 6.12 - Tabla de conexiones con tuplas

6. **[Equipo:PC1 - Usuario:dit]** Desde el navegador se accede a la web de trajano.us.es, para que en el equipo Encaminador se pueda buscar la entrada concreta de esta web.

7. **[Equipo:PC1 - Usuario:dit]** Para mostrar las entradas de conntrack en las cuales se pueda ver tráfico con un servidor web, se usa el siguiente comando.

El resultado de este comando se puede ver en la Figura 6.13.

Equipo:Encaminador **Usuario:**root

```
# -p indica el protocolo y --dport nos indica el puerto destino
# de la entrada
$ conntrack -L -p tcp --dport 80
```

```
root@localhost:/home/dit# conntrack -L -p tcp --dport 80
tcp      6 102 TIME_WAIT src=10.10.1.20 dst=193.147.162.130 sport=42802 dport=80 src=193.147.162.130 dst=192.168.106.128 sport=80 dport=42802 [ASSURED] mark=0 use=1
tcp      6 106 TIME_WAIT src=10.10.1.20 dst=193.147.162.130 sport=42806 dport=80 src=193.147.162.130 dst=192.168.106.128 sport=80 dport=42806 [ASSURED] mark=0 use=1
tcp      6 431991 ESTABLISHED src=10.10.1.20 dst=34.107.221.82 sport=44398 dport=80 src=34.107.221.82 dst=192.168.106.128 sport=80 dport=44398 [ASSURED] mark=0 use=1
tcp      6 106 TIME_WAIT src=10.10.1.20 dst=193.147.162.130 sport=42804 dport=80 src=193.147.162.130 dst=192.168.106.128 sport=80 dport=42804 [ASSURED] mark=0 use=1
tcp      6 431998 ESTABLISHED src=10.10.1.20 dst=93.184.220.29 sport=47372 dport=80 src=93.184.220.29 dst=192.168.106.128 sport=80 dport=47372 [ASSURED] mark=0 use=1
tcp      6 431991 ESTABLISHED src=10.10.1.20 dst=142.250.184.163 sport=35284 dport=80 src=142.250.184.163 dst=192.168.106.128 sport=80 dport=35284 [ASSURED] mark=0 use=1
tcp      6 106 TIME_WAIT src=10.10.1.20 dst=193.147.162.130 sport=42808 dport=80 src=193.147.162.130 dst=192.168.106.128 sport=80 dport=42808 [ASSURED] mark=0 use=1
tcp      6 431991 ESTABLISHED src=10.10.1.20 dst=34.107.221.82 sport=44392 dport=80 src=34.107.221.82 dst=192.168.106.128 sport=80 dport=44392 [ASSURED] mark=0 use=1
tcp      6 106 TIME_WAIT src=10.10.1.20 dst=193.147.162.130 sport=42810 dport=80 src=193.147.162.130 dst=192.168.106.128 sport=80 dport=42810 [ASSURED] mark=0 use=1
conntrack v1.4.6 (conntrack-tools): 9 flow entries have been shown.
root@localhost:/home/dit#
```

Figura 6.13 - Conexiones con servidores web

8. **[Equipo:PC1 - Usuario:dit]** Si se quiere ver aquellos paquetes cuya dirección destino original sea la de trajano (*193.147.162.130*), se usa el siguiente comando.

En la Figura 6.14, se puede observar la entrada de conntrack correspondiente a la comunicación con el servidor de trajano.

Equipo:Encaminador **Usuario:**root

```
$ conntrack -L -p tcp --dst 193.147.162.130
```

```
root@localhost:/home/dit# conntrack -L -p tcp --dst 193.147.162.130
tcp      6 117 TIME_WAIT src=10.10.1.20 dst=193.147.162.130 sport=42818 dport=80 src=193.147.162.130 dst=192.168.106.128 sport=80 dport=42818
```

Figura 6.14 - Entrada conexión con servidor trajano.us.es

Otra opción que permite conntrack, es colocarlo en forma (monitor de eventos), de forma que avise cada vez que una tupla sea introducida o eliminada.

9. **[Equipo:PC1 - Usuario:dit]** Si se desea, que conntrack nos avise de todos los eventos que se produzcan (Creación de entradas, actualización o destrucción de entradas), se usará el modo monitor con la opción -E.

Equipo:Encaminador **Usuario:**root

```
$ conntrack -E
```

De esta manera se puede observar los estados de los eventos de las entradas de conntrack (Figura 6.15).

```
conntrack v1.4.6 (conntrack-tools): 6 flow entries have been shown.
root@localhost:/home/dit# conntrack -E
[NEW] udp    17 30 src=10.10.1.20 dst=8.8.8.8 sport=57941 dport=53 [UNREPLIED] src=8.8.8.8 dst=192.168.106.128 sport=53 dport=57941
[NEW] tcp    6 120 SYN SENT src=10.10.1.20 dst=193.147.162.130 sport=42826 dport=80 [UNREPLIED] src=193.147.162.130 dst=192.168.106.128 sport=80 dport=42826
[UPDATE] udp   17 30 src=10.10.1.20 dst=8.8.8.8 sport=57941 dport=53 src=8.8.8.8 dst=192.168.106.128 sport=53 dport=57941
[NEW] tcp    6 120 SYN SENT src=10.10.1.20 dst=193.147.162.130 sport=42828 dport=80 [UNREPLIED] src=193.147.162.130 dst=192.168.106.128 sport=80 dport=42828
[UPDATE] tcp   6 60 SYN_RECV src=10.10.1.20 dst=193.147.162.130 sport=42826 dport=80 src=193.147.162.130 dst=192.168.106.128 sport=80 dport=42826
[UPDATE] tcp   6 432000 ESTABLISHED src=10.10.1.20 dst=193.147.162.130 sport=42826 dport=80 src=193.147.162.130 dst=192.168.106.128 sport=80 dport=42826 [ASSURED]
```

Figura 6.15 - Flujo de conexión con servidor trajano

Una vez se han generado las tuplas, comienza a contar su *time_out*, el cual se reiniciará cada vez que la tupla vea tráfico en alguna de las dos direcciones.

En caso de no ver tráfico en el periodo de *time_out*, se borrará la tupla.

```
[DESTROY] udp    17 src=10.10.1.20 dst=8.8.8.8 sport=57941 dport=53 src=8.8.8.8 dst=192.168.106.128 sport=53 dport=57941
[NEW]   udp    17 30 src=192.168.106.1 dst=224.0.0.251 sport=5353 dport=5353 [UNREPLIED] src=224.0.0.251 dst=192.168.106.1 sport=5353 dport=5353
[DESTROY] udp    17 src=192.168.106.1 dst=224.0.0.251 sport=5353 dport=5353 [UNREPLIED] src=224.0.0.251 dst=192.168.106.1 sport=5353 dport=5353
[NEW]   udp    17 30 src=192.168.106.128 dst=192.168.106.254 sport=68 dport=67 [UNREPLIED] src=192.168.106.254 dst=192.168.106.128 sport=67 dport=68
[UPDATE]  udp   17 30 src=192.168.106.128 dst=192.168.106.254 sport=68 dport=67 src=192.168.106.254 dst=192.168.106.128 sport=67 dport=68
[DESTROY]  tcp   6 src=10.10.1.20 dst=193.147.162.130 sport=42826 dport=80 src=193.147.162.130 dst=192.168.106.128 sport=80 dport=42826 [ASSURED]
[DESTROY]  udp   17 src=192.168.106.128 dst=192.168.106.254 sport=68 dport=67 src=192.168.106.254 dst=192.168.106.128 sport=67 dport=68
[DESTROY]  tcp   6 src=10.10.1.20 dst=193.147.162.130 sport=42834 dport=80 src=193.147.162.130 dst=192.168.106.128 sport=80 dport=42834 [ASSURED]
```

Figura 6.16 - Eliminación de tuplas

Vemos como, tras no ver tráfico en ninguna dirección y consumido el *time_out*, conntrack pasa a eliminar la entrada de ese flujo.

En la Figura 6.16, se ha remarcado la eliminación de la entrada de la consulta DNS de trajano y la entrada de conexión con el servidor del mismo.

10. **[Equipo:Encaminador - Usuario:dit]** Si deseamos saber, cuántos flujos de conexiones tenemos registrados, basta con usar la opción -C de conntrack, pudiendo comprobar su salida en la Figura 6.17.

```
Equipo: Encaminador Usuario: root
```

```
$ conntrack -C
$ conntrack -L
```

```
root@localhost:/home/dit# conntrack -C
1
root@localhost:/home/dit# conntrack -L
tcp      6 431723 ESTABLISHED src=10.10.1.20 dst=35.155.194.246 sport=44898 dport=443 src=35.155.194.246 dst=192.168.106.128
mark=0 use=1
conntrack v1.4.6 (conntrack-tools): 1 flow entries have been shown.
root@localhost:/home/dit#
```

Figura 6.17 - Mostrar el número de entradas que tiene conntrack

Conclusión del ejemplo:

Mediante este ejemplo, se ha mostrado las herramientas básicas para mostrar y entender las entradas mostradas por conntrack.

Así mismo, cabe destacar, que es importante el punto, en el que se indican, que los paquetes deben atravesar la estructura de nftables, ya que sin está conntrack no mostraría ninguna entrada. Son numerosos los *post* indicando que no se ven entradas en conntrack, indicando que la solución de estos se produce al crear la estructura de *hooks* de nftables.

También destacar, que debemos distinguir entre el modo monitor de conntrack y el listar las entradas de la misma. Ya que el primero, aporta información adicional acerca del estado de la entrada mostrada.

Una vez visto, como se representan los datos en una entrada de conntrack, así como los estados de las entradas en modo monitor y el uso de conntrack-tools, se pasará a estudiar los protocolos TCP, UDP e ICMP, viendo sus mensajes, la apertura de conexión y los estados de cada protocolo.

6.3 TCP Conntrack

En este punto, se estudiará el protocolo TCP en conntrack.

Se recomienda la revisión del anexo de protocolos, en especial, el punto “Protocolo TCP (Transmission Control Protocol)” [49].

TCP, es un protocolo orientado a conexión, es decir, antes de realizar el intercambio de datos es necesario el establecimiento de una conexión, para lo cual, TCP usa un triple *handshake*.

Al finalizar la transmisión de datos, la conexión se cierra de forma muy parecida a como se realiza la apertura, de forma que se procede a ver los mensajes que estos usan y las fases de conexión de TCP.

Al igual que se han visto dos tipos de estados en el punto “Máquina de estados.”, existe otro tipo más, los estados de TCP.

Estos estados, a diferencia de los de la máquina de estados de conntrack, no se usan para marcar los paquetes, sino, que nos dan información del estado de la comunicación TCP. Los estados internos que conntrack usa para TCP, se encuentran definidos en *linux/include/uapi/linux/netfilter/nf_conntrack_tcp.h* [50] y son los vistos en la Figura 6.18.

```
8 /* This is exposed to userspace (ctnetlink) */
9 enum tcp_conntrack {
10     TCP_CONNTRACK_NONE,
11     TCP_CONNTRACK_SYN_SENT,
12     TCP_CONNTRACK_SYN_RECV,
13     TCP_CONNTRACK_ESTABLISHED,
14     TCP_CONNTRACK_FIN_WAIT,
15     TCP_CONNTRACK_CLOSE_WAIT,
16     TCP_CONNTRACK_LAST_ACK,
17     TCP_CONNTRACK_TIME_WAIT,
18     TCP_CONNTRACK_CLOSE,
19     TCP_CONNTRACK_LISTEN /* obsolete */
20 #define TCP_CONNTRACK_SYN_SENT2 TCP_CONNTRACK_LISTEN
```

Figura 6.18 - Estados TCP definidos en el código de Netfilter

Estos estados, tienen un *timeout* por defecto, que es el tiempo en el que una entrada puede permanecer en ese estado antes de ser eliminada, en la Tabla 14 se recogen los estados y su tiempo por defecto.

Estado	Timeout
<i>NONE</i>	30 minutos
<i>SYN_SENT</i>	2 minutos
<i>SYN_RECV</i>	60 segundos
<i>ESTABLISHED</i>	5 días
<i>FIN_WAIT</i>	2 minutos
<i>CLOSE_WAIT</i>	12 horas
<i>LAST_ACK</i>	30 segundos
<i>TIME_WAIT</i>	2 minutos
<i>CLOSE</i>	10 segundos

Tabla 14 - Valores por defecto estados TCP

6.3.1 Ejemplo conntrack TCP.

Descripción del ejemplo:

En este ejemplo, se realizará una conexión TCP completa, a través de la cual, se analizarán los diferentes paquetes enviados y su repercusión en el módulo conntrack.

Se verá las entradas creadas, la monitorización de estas y el camino del paquete a través de la estructura de cadenas y reglas de nftables.

Para este ejemplo, se usará el escenario de 2 equipos, “Anexo E - Escenario 2 Equipos”.

- Encaminador, será el equipo que tendrá nftables y desde el que se verá el seguimiento de conexiones
- PC1, será el encargado de generar tráfico, usando las herramientas; hping3 [51] y curl [52].

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Para el ejemplo de TCP, se crea una tabla filter, una tabla nat y otra netdev.

```
Equipo: Encaminador Usuario:root

$ nft add table ip nat
$ nft add table ip filter
$ nft add table netdev filteringress
```

2. **[Equipo:Encaminador - Usuario:root]** Se crean los siguientes grupos de cadenas:

- a. **Tabla filter:** cadenas prerouting, input, forward, postrouting y output.
- b. **Tabla nat:** cadenas prerouting y postrouting.
- c. **Tabla netdev:** cadena filteringress.

```
Equipo: Encaminador Usuario:root

$ nft add chain 'ip nat prerouting { type nat hook prerouting priority -1 ; }'
$ nft add chain 'ip nat postrouting { type nat hook postrouting priority -1 ; }'
$ nft add chain 'ip filter prerouting { type filter hook prerouting priority -1 ; }'
$ nft add chain 'ip filter postrouting { type filter hook postrouting priority -1 ; }'
$ nft add chain 'ip filter input { type filter hook input priority -1 ; }'
$ nft add chain 'ip filter forward { type filter hook forward priority 0 ; }'
$ nft add chain 'ip filter output { type filter hook output priority -1 ; }'
$ nft add chain 'ip filter prerouting { type filter hook prerouting priority -1 ; }'
$ nft add 'chain netdev filteringress ingress { type filter hook ingress device ens38 priority 0 ; }'
```

3. [Equipo:Encaminador - Usuario:root] Se crean las siguientes reglas:

- a. En la cadena filteringress, se añade una regla que active *nftrace* para todo el tráfico entrante.
- b. En la cadena prerouting de filter, se añade una regla que contabilice el tráfico TCP cuyo *state* sea *invalid*.
- c. En la cadena prerouting de filter, se añade una regla que contabilice el tráfico TCP cuyo estado sea *new* o *established*.
- d. En la cadena output de filter, se añade una regla que active *nftrace* para todo el tráfico saliente.

Equipo: Encaminador **Usuario:** root

```
$ nft add rule netdev filteringress nftrace set 1
$ nft add rule ip filter prerouting ip protocol tcp ct state invalid counter continue
$ nft add rule ip filter prerouting ip protocol tcp ct state { new, established } counter continue
$ nft add rule ip filter output nftrace set 1
```

4. [Equipo:Encaminador - Usuario:root] Se muestra el *ruleset* de nftables y se comprueba que su salida sea la observada en la Figura 6.19.

Equipo: Encaminador **Usuario:** root

```
$ nft list ruleset
```

```
root@localhost:/home/dit# nft list ruleset
table ip nat {
    chain postrouting {
        type nat hook postrouting priority -1; policy accept;
    }

    chain prerouting {
        type nat hook prerouting priority -1; policy accept;
    }
}
table ip filter {
    chain prerouting {
        type filter hook prerouting priority -1; policy accept;
        ip protocol tcp ct state invalid counter packets 0 bytes 0 continue
        ip protocol tcp ct state { established, new } counter packets 0 bytes 0 continue
    }

    chain postrouting {
        type filter hook postrouting priority -1; policy accept;
    }

    chain output {
        type filter hook output priority -1; policy accept;
        nftrace set 1
    }

    chain input {
        type filter hook input priority -1; policy accept;
    }
}
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
    }
}
table netdev filteringress {
    chain ingress {
        type filter hook ingress device ens38 priority 0; policy accept;
        nftrace set 1
    }
}
```

Figura 6.19 - Estructura TCP conntrack

5. **[Equipo:Encaminador - Usuario:root]** En Encaminador, se abren dos terminales adicionales con el usuario root:

- En el primero de ellos, se realizará la monitorización del paso de los paquetes por las reglas de nftables
- En el segundo de ellos, colocaremos conntrack en modo evento, de forma que muestre todos los eventos que se produzcan durante la comunicación.

```
Equipo:Encaminador Usuario:root
```

```
#Terminal 1
$ nft monitor trace

#Terminal 2
$ conntrack -E --event-mask ALL
```

6. **[Equipo:PC1 - Usuario:root]** Desde PC1, se usa el siguiente comando para pedir la página web del servidor Apache de Encaminador.

```
Equipo:PC1 Usuario:root
```

```
$ curl 10.10.1.10
```

7. **[Equipo:Encaminador - Usuario:root]** Se observa la salida mostrada en el terminal con el monitor activado, en el cual, se puede ver el camino del paquete TCP que llega al equipo Encaminador por la interfaz “ens38”, este paquete tiene activado el *flag* SYN.

```
trace id 1f910109 ip nat postouting
trace id 1f910109 netdev filteringress ingress packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dsrp cs0 ip ecn not-ect ip ttl 64 ip id 45985 ip length 60 tcp sport 38340 tcp dport http tcp flags == syn tcp window 29200
trace id 1f910109 netdev filteringress ingress rule nftrace set 1 (verdict continue)
trace id 1f910109 netdev filteringress ingress verdict continue
trace id 1f910109 netdev filteringress ingress
trace id 1f910109 ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dsrp cs0 ip ecn not-ect ip ttl 64 ip id 45985 ip length 60 tcp sport 38340 tcp dport http tcp flags == syn tcp window 29200
trace id 1f910109 ip filter prerouting rule ip protocol tcp ct state { } counter packets 0 bytes 0 continue (verdict continue)
trace id 1f910109 ip filter prerouting verdict continue
trace id 1f910109 ip filter prerouting
trace id 1f910109 ip nat prerouting verdict continue
trace id 1f910109 ip nat prerouting
trace id 1f910109 ip filter input verdict continue
trace id 1f910109 ip filter input
trace id 1f910109 inet filter input verdict continue
trace id 1f910109 inet filter input
```

Figura 6.20 - Camino paquete TCP SYN

En la Figura 6.20, se puede observar el camino del paquete TCP SYN.

Camino del paquete TCP SYN:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena filteringress, la cual, activa la marca de seguimiento del paquete.
- 3) El veredicto en filteringress es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter, (este pasa por la cadena nat pero no se le aplica ninguna regla).
- 4) En prerouting, hace *match* con la regla que realiza la cuenta de paquetes, cuyo estado es *new*

(En el punto anterior de este capítulo , se vio que la categorización de este estado tiene lugar en este *hook*, en una etapa temprana de procesado).

- 5) El veredicto del paquete es *continue*, por lo que se procede a consultar el encaminamiento del mismo.
 - 6) Al ser un paquete cuyo destino es el proceso local, continua su camino hacia el *hook input*.
 - 7) Antes de salir del *hook input* y siendo su veredicto *continue*, se crea la entrada con el estado NEW en conntrack.
8. [Equipo:Encaminador - Usuario:root] Se observa la salida mostrada en el terminal con el monitor activado, en este, se puede ver el camino del paquete TCP generado por el proceso local, el cual responde con un paquete SYN-ACK que sale por la interfaz “ens38” como se puede ver en la Figura 6.21.

```
trace id 850cf6d5 ip filter output packet: oif "ens38" ip saddr 10.10.1.10 ip daddr 10.10.1.20 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 0 ip length 60 tcp sport http tcp dport 38340 tcp flags == 0x12 tcp window 65160
trace id 850cf6d5 ip filter output rule nftrace set 1 (verdict continue)
trace id 850cf6d5 ip filter output verdict continue
trace id 850cf6d5 ip filter output
trace id 850cf6d5 inet filter output verdict continue
trace id 850cf6d5 inet filter output
trace id 850cf6d5 ip filter postrouting verdict continue
trace id 850cf6d5 ip filter postrouting
```

Figura 6.21 - Camino paquete TCP SYN-ACK

Camino del paquete TCP SYN-ACK:

- 1) El paquete TCP SYN ACK es generado por el proceso local y antes de llegar a output se consulta la tabla de encaminamiento.
 - 2) Tras comprobar su encaminamiento, el paquete llega al *hook output* donde se le activa *nftrace*.
 - 3) El veredicto de output es *continue* y el paquete sigue su curso hacia postrouting
 - 4) Antes de salir del *hook postrouting* se actualiza la entrada que el paquete TCP SYN había creado en conntrack, siendo el nuevo estado de la entrada UPDATE.
9. [Equipo:Encaminador - Usuario:root] Por último, PC1 responde con un paquete TCP con el flag ACK activado y de esta manera queda establecida la conexión TCP.

```
trace id c49e8d16 netdev filteringress ingress packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether dad dr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 45986 ip length 52 tcp sport 38340 tcp dport http tcp flags == ack tcp window 229
trace id c49e8d16 netdev filteringress ingress rule nftrace set 1 (verdict continue)
trace id c49e8d16 netdev filteringress ingress verdict continue
trace id c49e8d16 netdev filteringress ingress
trace id c49e8d16 ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 45986 ip length 52 tcp sport 38340 tcp dport http tcp flags == ack tcp window 229
trace id c49e8d16 ip filter prerouting rule ip protocol tcp ct state { } counter packets 0 bytes 0 continue (verdict continue)
trace id c49e8d16 ip filter prerouting verdict continue
trace id c49e8d16 ip filter prerouting
trace id c49e8d16 ip filter input verdict continue
trace id c49e8d16 ip filter input
trace id c49e8d16 inet filter input verdict continue
trace id c49e8d16 inet filter input
```

Figura 6.22 - Camino paquete TCP ACK

Camino del paquete TCP ACK:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena filteringress, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en filteringress es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 4) En prerouting, hace *match* con la regla que realiza la cuenta de paquetes cuyo estado es *established*.
- 5) El veredicto del paquete es *continue*, por lo que se procede a consultar el encaminamiento del mismo.
- 6) Al ser un paquete, cuyo destino es el proceso local, continua su camino hacia el *hook input*.
- 7) Antes de salir del *hook input* y siendo su veredicto *continue*, se actualiza la entrada con el estado UPDATE en conntrack.

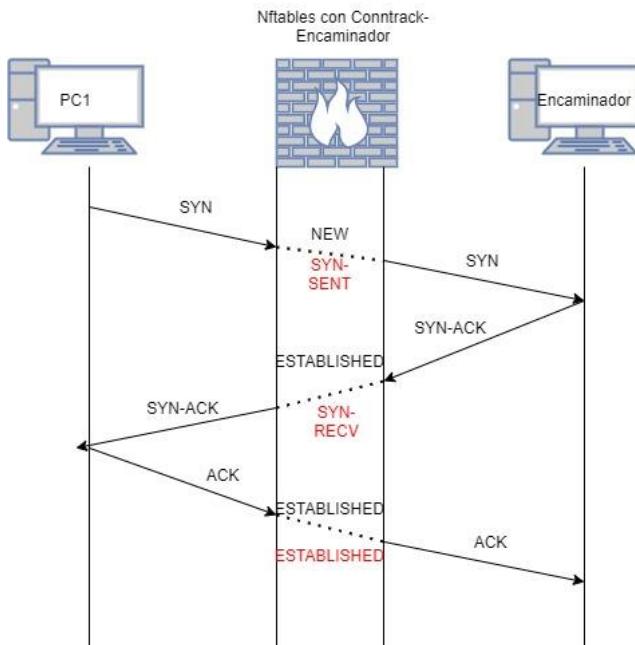


Figura 6.23 - Esquema estados conntrack TCP handshake

En la Figura 6.23, se puede observar un esquema del transcurso de los paquetes de apertura de conexión TCP y los distintos estados que estos toman.

Los estados en negro, pertenecen a la categorización que realiza conntrack en el *hook prerouting* y *hook output*, Tabla 9, mientras que aquellos que aparecen en color rojo, son los estados internos de TCP que se ven en la Figura 6.18, los cuales, adjudica conntrack en los puntos de creación/actualización de entradas en el *hook input* y *hook postrouting*.

Este es el transcurso del paquete por las diferentes reglas, pero el paso por el *hook input* y el *hook postrouting*, actualizan el sistema de seguimiento de conexiones, produciendo que;

- 1) Al llegar el paquete SYN, se añade una nueva entrada con el estado interno de TCP **SYN_SENT** (línea 1 Figura 6.24).
- 2) Cuando el proceso local de Encaminador recibe el paquete SYN, este envía un mensaje de respuesta a PC1 con el SYN-ACK, al cruzar este mensaje por el *hook postrouting*, se actualiza la entrada de conntrack con el estado interno de TCP **SYN_RECV**, además, al haber visto

tráfico ya en ambos sentidos, se pone a 0 el valor del flag UNREPLIED de la entrada (línea 2 Figura 6.24).

- 3) Por último, cuando PC1 recibe este paquete, contestará con el ACK visto en la Figura 6.22, de forma de que al llegar este paquete al *hook input*, se volvería a actualizar la entrada de conntrack y el valor de su estado interno pasaría a ser ESTABLISHED.

Se activaría el flag de ASSURED, puesto que la conexión estaría establecida y en caso de que se llenara la memoria de conntrack, no se busca que se borre esa entrada, por lo que se marca como asegurada (línea 3 Figura 6.24).

```
[NEW] tcp      6 120 SYN SENT src=10.10.1.20 dst=10.10.1.10 sport=38340 dport=80 [UNREPLIED] src=10.10.1.10 dst=10.10.1.20 sport=80 dport=38340
[UPDATE] tcp    6 60 SYN_RECV src=10.10.1.20 dst=10.10.1.10 sport=38340 dport=80 src=10.10.1.10 dst=10.10.1.20 sport=80 dport=38340
[UPDATE] tcp    6 432000 ESTABLISHED src=10.10.1.20 dst=10.10.1.10 sport=38340 dport=80 src=10.10.1.10 dst=10.10.1.20 sport=80 dport=38340 [ASSURED]
```

Figura 6.24 - Entradas conntrack paquete TCP

En la Figura 6.25 se puede ver el flow del handshake de TCP descrito en este ejemplo.

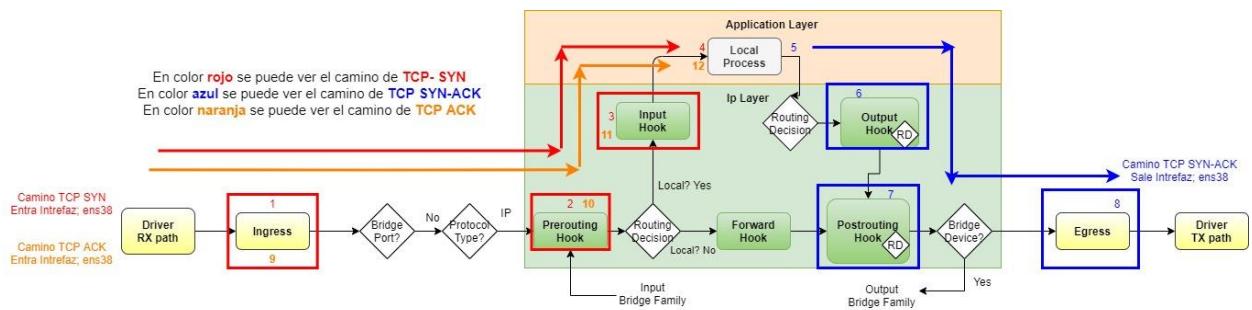


Figura 6.25 - Flow ejemplo TCP handshake conntrack

A continuación, se analizará dentro del mismo ejemplo el cierre de la conexión que tiene lugar una vez la web ha sido servida.

Se sigue viendo la salida de los dos terminales abiertos en el equipo Encaminador.

El cierre de conexión se realiza mediante el intercambio de 3 mensajes (muy parecido al handshake).

10. **[Equipo:Encaminador - Usuario:root]** Se observa la salida mostrada en el terminal con el monitor activado, en el cual, se puede ver el camino del paquete TCP que llega al equipo Encaminador por la interfaz “ens38”, este paquete tiene activado los flags FIN y ACK (Figura 6.26).

```
trace id 94140c78 ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 5675 ip length 52 tcp sport 38342 tcp dport http tcp flags == 0x11 tcp window 403
trace id 94140c78 ip filter prerouting rule ip protocol tcp ct state { } counter packets 6 bytes 394 continue (verdict continue)
trace id 94140c78 ip filter prerouting verdict continue
trace id 94140c78 ip filter prerouting
trace id 94140c78 ip filter input verdict continue
trace id 94140c78 ip filter input
trace id 94140c78 inet filter input verdict continue
trace id 94140c78 inet filter input
```

Figura 6.26 - Camino paquete TCP FIN-ACK PC1

Camino del paquete TCP FIN:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”.
 - 2) Tras realizar las comprobaciones básicas del mismo, la NIC pasa el paquete a la cadena filteringress, la cual activa la marca de seguimiento del paquete.
 - 3) El veredicto en filteringress es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
 - 4) En prerouting, hace *match* con la regla que realiza la cuenta de paquetes cuyo estado es *established*.
 - 5) El veredicto del paquete es *continue*, por lo que se procede a consultar el encaminamiento del mismo.
 - 6) Al ser un paquete cuyo destino es el proceso local, continua su camino hacia el *hook input*.
 - 7) Antes de salir del *hook input* y siendo su veredicto *continue*, se actualiza la entrada con el estado UPDATE en conntrack, segunda línea de la Figura 6.29.
11. [Equipo:Encaminador - Usuario:root] Se observa la salida mostrada en el terminal con el monitor activado, en el cual se puede ver el camino del paquete TCP generado por el proceso local, el cual responde con un paquete FIN-ACK que sale por la interfaz “ens38” como se puede ver en la Figura 6.27.

```
trace id 12568ad1 ip filter output packet: oif "ens38" ip saddr 10.10.1.10 ip daddr 10.10.1.20 ip dscp cs0
ip ecn not-ect ip ttl 64 ip id 48547 ip length 52 tcp sport http tcp dport 38342 tcp flags == 0x11 tcp wi
ndow 509
trace id 12568ad1 ip filter output rule nftrace set 1 (verdict continue)
trace id 12568ad1 ip filter output verdict continue
trace id 12568ad1 ip filter output
trace id 12568ad1 inet filter output verdict continue
trace id 12568ad1 inet filter output
trace id 12568ad1 ip filter postrouting verdict continue
trace id 12568ad1 ip filter postrouting
```

Figura 6.27 - Camino paquete TCP FIN-ACK Encaminador

Camino del paquete TCP FIN-ACK:

- 1) El paquete TCP FIN ACK es generado por el proceso local y antes de llegar a output se consulta la tabla de encaminamiento.
 - 2) Tras comprobar su encaminamiento, el paquete llega al *hook output* donde se activa *nftrace*.
 - 3) El veredicto de output es *continue* y el paquete sigue su curso hacia postrouting
 - 4) Antes de salir del *hook postrouting*, se actualiza la entrada que el paquete TCP FIN había creado en conntrack, siendo el nuevo estado de la entrada UPDATE, tercera línea de la Figura 6.29.
12. [Equipo:Encaminador - Usuario:root] Por último, PC1 responde con un paquete TCP con el flag ACK activado y de esta manera queda cerrada la conexión TCP.

```

trace id 740c489c netdev filteringress ingress packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether dad
dr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 56
76 ip length 52 tcp sport 38342 tcp dport http tcp flags == ack tcp window 403
trace id 740c489c netdev filteringress ingress rule nftrace set 1 (verdict continue)
trace id 740c489c netdev filteringress ingress verdict continue
trace id 740c489c netdev filteringress ingress
trace id 740c489c ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c
:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 5676 ip le
ngth 52 tcp sport 38342 tcp dport http tcp flags == ack tcp window 403
trace id 740c489c ip filter prerouting rule ip protocol tcp ct state { } counter packets 6 bytes 394 cont
inue (verdict continue)
trace id 740c489c ip filter prerouting verdict continue
trace id 740c489c ip filter prerouting
trace id 740c489c ip filter input verdict continue
trace id 740c489c ip filter input
trace id 740c489c inet filter input verdict continue
trace id 740c489c inet filter input

```

Figura 6.28 - Camino paquete TCP ACK cierre conexión

Camino del paquete TCP ACK:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC pasa el paquete a la cadena filteringress, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en filteringress es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 4) En prerouting, hace *match* con la regla que realiza la cuenta de paquetes cuyo estado es *established*.
- 5) El veredicto del paquete es *continue*, por lo que se procede a consultar el encaminamiento del mismo.
- 6) Al ser un paquete cuyo destino es el proceso local, continua su camino hacia el *hook input*.
- 7) Antes de salir del *hook input* y siendo su veredicto *continue*, se actualiza la entrada con el estado UPDATE en conntrack, cuarta línea de la Figura 6.29

```

[UPDATE] tcp      6 432000 ESTABLISHED src=10.10.1.20 dst=10.10.1.10 sport=38340 dport=80 src=10.10.1.10 dst=10.10.1.20 sport=80 dport=38340 [ASSURED]
[UPDATE] tcp      6 120 FIN_WAIT src=10.10.1.20 dst=10.10.1.10 sport=38340 dport=80 src=10.10.1.10 dst=10.10.1.20 sport=80 dport=38340 [ASSURED]
[UPDATE] tcp      6 30 LAST ACK src=10.10.1.20 dst=10.10.1.10 sport=38340 dport=80 src=10.10.1.10 dst=10.10.1.20 sport=80 dport=38340 [ASSURED]
[UPDATE] tcp      6 120 TIME_WAIT src=10.10.1.20 dst=10.10.1.10 sport=38340 dport=80 src=10.10.1.10 dst=10.10.1.20 sport=80 dport=38340 [ASSURED]

```

Figura 6.29 - Entradas conntrack cierre conexión TCP

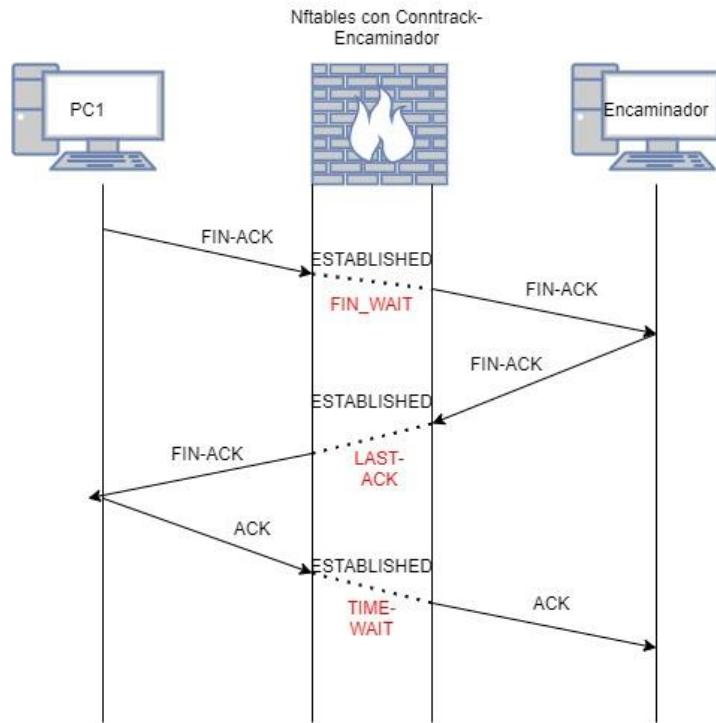


Figura 6.30 - Esquema estados conntrack TCP close connection

En la Figura 6.30, se puede observar un esquema del transcurso de los paquetes para cerrar la conexión TCP y los distintos estados que estos toman.

Los estados en negro, pertenecen a la categorización que realiza conntrack en el *hook prerouting* y *hook output*, Tabla 9, mientras que aquellos que aparecen en color rojo, son los estados internos de TCP que se ven en la Figura 6.18, los cuales adjudica conntrack en los puntos de creación/actualización de entradas en el *hook input* y *hook postrouting*.

- 1) Podemos ver, como la entrada tenía como estado ESTABLISHED, de forma que al enviar PC1 el paquete con el flag FIN y al pasar este por el *hook input*, la entrada se actualiza a su estado FIN_WAIT, el cual indica que se espera el cierre de la conexión.
- 2) Encaminador envía su paquete TCP con los flags FIN y ACK, al pasar este por el *hook postrouting*, se actualiza la entrada con el estado LAST_ACK, indicando que lo último que falta para cerrar la conexión es recibir el ACK del cliente, de forma que al recibir este paquete, PC1 enviará el último ACK para cerrar la conexión.
- 3) Al llegar este paquete al *hook input*, se actualiza la entrada y su estado pasa a ser TIME_WAIT.

Este estado tiene una duración por defecto de 2 min, por lo que este tiempo se emplea para que todos los paquetes que han tenido problemas y que han sufrido algún tipo de retraso o que se encuentren en un buffer de procesamiento, puedan atravesar igualmente el conjunto de reglas, incluso después de que la conexión se haya cerrado.

Este tiempo se puede considerar como una especie de “tiempo de guarda”, que permite que todos los paquetes lleguen a su destino de forma correcta en caso de que la red tenga algún tipo de congestión.

En la Figura 6.31, se puede ver el flow del cierre de conexión TCP que se ha descrito en este ejemplo.

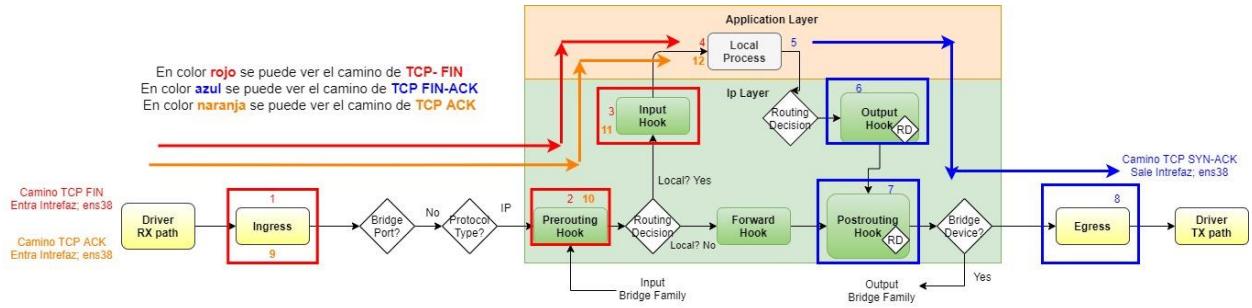


Figura 6.31 - Flow ejemplo TCP cierre conexión conntrack

13. [Equipo:Encaminador - Usuario:root] Si se intenta ver las entradas de conntrack, veremos que solo se encuentra la entrada con el estado TIME_WAIT y si se lanza varias veces la consulta, se puede ver como el temporizador va decrementándose, hasta que la entrada desaparece (Figura 6.32).

Equipo: Encaminador Usuario: root

```
$ conntrack -L
# Se espera un par de segundos
$ conntrack -L
# Se espera un par de segundos
$ conntrack -L
```

```
root@localhost:/home/dit# conntrack -L
tcp 6 4 TIME_WAIT src=10.10.1.20 dst=10.10.1.10 sport=38342 dport=80 src=10.10.1.10 dst=10.10.1.20 sport=80 dport=38342 [ASSURED] mark=0 use=1
udp 17 17 src=192.168.106.1 dst=224.0.0.251 sport=5353 dport=5353 [UNREPLIED] src=224.0.0.251 dst=192.168.106.1 sport=5353 dport=5353 mark=0 use=1
conntrack v1.4.6 (conntrack-tools): 2 flow entries have been shown.
root@localhost:/home/dit# conntrack -L
tcp 6 1 TIME_WAIT src=10.10.1.20 dst=10.10.1.10 sport=38342 dport=80 src=10.10.1.10 dst=10.10.1.20 sport=80 dport=38342 [ASSURED] mark=0 use=1
udp 17 14 src=192.168.106.1 dst=224.0.0.251 sport=5353 dport=5353 [UNREPLIED] src=224.0.0.251 dst=192.168.106.1 sport=5353 dport=5353 mark=0 use=1
conntrack v1.4.6 (conntrack-tools): 2 flow entries have been shown.
root@localhost:/home/dit# conntrack -L
tcp 6 0 TIME_WAIT src=10.10.1.20 dst=10.10.1.10 sport=38342 dport=80 src=10.10.1.10 dst=10.10.1.20 sport=80 dport=38342 [ASSURED] mark=0 use=1
udp 17 13 src=192.168.106.1 dst=224.0.0.251 sport=5353 dport=5353 [UNREPLIED] src=224.0.0.251 dst=192.168.106.1 sport=5353 dport=5353 mark=0 use=1
conntrack v1.4.6 (conntrack-tools): 1 flow entries have been shown.
root@localhost:/home/dit# conntrack -L
tcp 6 12 src=192.168.106.1 dst=216.58.215.131 sport=5353 dport=5353 [UNREPLIED] src=224.0.0.251 dst=192.168.106.1 sport=5353 dport=5353 mark=0 use=1
conntrack v1.4.6 (conntrack-tools): 1 flow entries have been shown.
```

Figura 6.32 - Envejecimiento entrada conntrack

Debemos hacer una puntuación en este ejemplo ya que tiene una excepción de estados.

- En este ejemplo, el paquete FIN-ACK, ha llegado por la cadena input, de forma que los estados que se usarán para el cierre de la conexión son: (FIN_WAIT, LAST_ACK, TIME_WAIT).
- En caso de que el paquete FIN_ACK, sea visto primero por el hook postrouting, los estados serían los vistos en la Figura 6.33, (FIN_WAIT, CLOSE_WAIT, LAST_ACK, TIME_WAIT).

```
[UPDATE] tcp 6 432000 ESTABLISHED src=192.168.106.137 dst=216.58.215.131 sport=43896 dport=80 src=216.58.215.131 dst=192.168.106.137 sport=80 dport=43896
[UPDATE] tcp 6 120 FIN_WAIT src=192.168.106.137 dst=216.58.215.131 sport=43896 dport=80 src=216.58.215.131 dst=192.168.106.137 sport=80 dport=43896
[UPDATE] tcp 6 60 CLOSE_WAIT src=192.168.106.137 dst=216.58.215.131 sport=43896 dport=80 src=216.58.215.131 dst=192.168.106.137 sport=80 dport=43896
[UPDATE] tcp 6 30 LAST_ACK src=192.168.106.137 dst=216.58.215.131 sport=43896 dport=80 src=216.58.215.131 dst=192.168.106.137 sport=80 dport=43896
[UPDATE] tcp 6 120 TIME_WAIT src=192.168.106.137 dst=216.58.215.131 sport=43896 dport=80 src=216.58.215.131 dst=192.168.106.137 sport=80 dport=43896
```

Figura 6.33 - Entradas conntrack del cierre de conexión del Encaminador

Conclusión del ejemplo:

De este ejemplo, se pueden obtener dos imágenes muy importantes, la Figura 6.23 y la Figura 6.30, las cuales muestran el proceso de apertura y cierre de una conexión TCP, indicando los estados TCP, así como la categorización que se realiza de los paquetes usados en este proceso.

Estos ejemplos tienen especial importancia para un administrador de red, ya que muestran de forma

clara y detallada, el camino seguido por cada paquete y los cambios producidos en los estados de las entradas conntrack.

Debemos tener en cuenta, que conntrack, realiza un mantenimiento de estados de los equipos finales, es decir, no es capaz de mantener una entrada con el estado del emisor y otra con la del receptor, sino que la misma entrada presenta los cambios y los estados en los que se encuentra la comunicación de ambos equipos finales.

Sería un buen punto de trabajo, para el futuro, que este mantenimiento de estados se realizase de forma independiente según emisor y receptor, ya que, en casos específicos, podemos encontrarnos con que el estado de uno de ellos, difiera del otro, esto puede ser producido por pérdidas, retrasos de la red etc...

Como se ha mencionado anteriormente, en muchos escritos se puede ver que el primer paquete que atraviesa nftables, es el que crea la entrada de conntrack y esto no es así.

Para que conntrack cree una entrada de TCP, el paquete que este debe ver es de apertura de conexión, en caso contrario, considerará que el estado de ese paquete es INVALID, puesto que no puede crear una entrada al no ser un paquete de apertura de conexión y tampoco se puede asociar a un flujo existente, por lo que se categoriza como inválido.

6.3.2 Ejemplo conntrack TCP invalid.

Descripción del Ejemplo:

Desde PC1, probaremos a generar paquetes TCP de distintos tipos.

En Encaminador, se activará la monitorización de eventos TCP, viendo así, cuáles de ellos son capaces de crear una entrada en conntrack, además se usarán reglas para poder ver que paquetes se categorizan como inválidos.

Para este ejemplo, se parte de la base creada en el ejemplo anterior, el cual usa el escenario de 2 equipos “Anexo E - Escenario 2 Equipos”.

Análisis del ejemplo:

1. [Equipo:Encaminador - Usuario:root] Para el ejemplo de TCP, se crea una tabla filter y otra ingress.

```
Equipo: Encaminador Usuario:root

$ nft add table ip filter
$ nft add table netdev ingress
```

2. [Equipo:Encaminador - Usuario:root] Se crean los siguientes grupos de cadenas:

- 1) Tabla filter: cadenas prerouting, input, forward, postrouting y output.
- 2) Tabla ingress: cadena ingr

```
Equipo: Encaminador Usuario:root

$ nft add chain 'ip filter prerouting { type filter hook prerouting priority 0 ; }'
$ nft add chain 'ip filter postrouting { type filter hook postrouting priority 0 ; }'
$ nft add chain 'ip filter input { type filter hook input priority 0 ; }'
```

```

$ nft add chain 'ip filter forward { type filter hook forward
priority 0 ; }'
$ nft add chain 'ip filter output { type filter hook output
priority 0 ; }'

$ nft add 'chain netdev ingress ingr { type filter hook ingress
device ens38 priority 0 ; }'

```

3. [Equipo:Encaminador - Usuario:root] Se crean las siguientes reglas:

- 1) Regla en la cadena prerouting de filter, para paquetes TCP cuyo *state* sea *invalid*.
- 2) Regla en la cadena prerouting de filter, para paquetes TCP cuyo *state* sea *new*.
- 3) Regla en la cadena prerouting de filter, para paquetes TCP cuyo *state* sea *established*.
- 4) Regla en la cadena ingr de ingress, para activar la marca *nftrace* de los paquetes TCP.

Equipo: Encaminador **Usuario:**root

```

#Reglas tabla filter
$ nft add rule ip filter prerouting ip protocol tcp ct state
invalid counter continue
$ nft add rule ip filter prerouting ip protocol tcp ct state new
counter continue
$ nft add rule ip filter prerouting ip protocol tcp ct state
established counter continue

#Regla tabla netdev
$ nft add rule netdev ingress ip protocol tcp nftrace set 1

```

4. [Equipo:Encaminador - Usuario:root] En el equipo Encaminador, se abre un terminal y se coloca conntrack en modo monitor, usando la opción para mostrar todos los eventos.

En otro terminal se abre el monitor de *nftrace*.

Equipo:Encaminador **Usuario:**root

```

#Terminal 1
$ conntrack -E --event-mask ALL

#Terminal 2
$ nft monitor trace

```

5. [Equipo:PC1 - Usuario:root] Desde PC1, se realiza el envío de diferentes paquetes TCP con el comando hping3.

Equipo:PC1 **Usuario:**root

```

#Flag PUSH
$ hping3 -c 1 -P 10.10.1.10 -p 80

#Flag URGENT
$ hping3 -c 1 -U 10.10.1.10 -p 80

#Flag RESET

```

```

$ hping3 -c 1 -R 10.10.1.10 -p 80
#Flag FIN
$ hping3 -c 1 -F 10.10.1.10 -p 80
#Flag ACK
$ hping3 -c 1 -A 10.10.1.10 -p 80
#Flag SYN
$ hping3 -c 1 -S 10.10.1.10 -p 80
#Flag SYN-ACK
$ hping3 -c 1 -S -A 10.10.1.10 -p 80

```

Se verá ahora el camino de cada uno de estos paquetes, para ver si abren una entrada en conntrack o si por el contrario son categorizados como *invalid*.

6. [Equipo:Encaminador - Usuario:root] En Encaminador se observa el monitor de nftrace para ver el transcurso de los distintos paquetes generados por PC1.

En este punto se analizarán todos los caminos de los diferentes paquetes TCP.

1) TCP PSH

```

trace id 0ec0bb21 netdev ingress ingr rule ip protocol tcp nftrace set 1 counter packets 2 bytes 92 (verdict continue)
trace id 0ec0bb21 netdev ingress ingr verdict continue
trace id 0ec0bb21 netdev ingress ingr
trace id 0ec0bb21 ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 27476 ip length 40 tcp sport 1357 tcp dport 0 tcp flags == psh tcp window 512
trace id 0ec0bb21 ip filter prerouting rule ip protocol tcp ct state invalid counter packets 0 bytes 0 (verdict continue)
trace id 0ec0bb21 ip filter prerouting verdict continue
trace id 0ec0bb21 ip filter prerouting
trace id 0ec0bb21 ip filter input verdict continue
trace id 0ec0bb21 ip filter input

```

Figura 6.34 - Camino TCP PSH

En la Figura 6.34, se puede ver el camino del paquete TCP PSH:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC pasa el paquete a la cadena *ingress*, la cual activa la marca de seguimiento del paquete.
- 3) En la cadena *ingress* se hace *match* con la primera regla y se activa la traza *nftrace* del paquete, su veredicto en esta cadena es *continue*.
- 4) En *prerouting*, hace *match* con la regla que comprueba el *state invalid* de los paquetes TCP.

Este paquete TCP tiene activado el flag PSH.

Al ser su estado inválido, este paquete no es capaz de crear una entrada en conntrack.

2) TCP URG

```
trace id c264da93 netdev ingress ingr rule ip protocol tcp nftrace set 1 counter packets 2 bytes 92 (verdict continue)
trace id c264da93 netdev ingress ingr verdict continue
trace id c264da93 netdev ingress ingr
trace id c264da93 ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 28465 ip length 40 tcp sport 2212 tcp dport 0 tcp flags == urg tcp window 512
trace id c264da93 ip filter prerouting rule ip protocol tcp ct state invalid counter packets 0 bytes 0 (verdict continue)
trace id c264da93 ip filter prerouting verdict continue
trace id c264da93 ip filter prerouting
trace id c264da93 ip filter input verdict continue
trace id c264da93 ip filter input
```

Figura 6.35 - Camino TCP URG

En la Figura 6.35, se puede ver el camino del paquete TCP URG:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC pasa el paquete a la cadena ingress, la cual activa la marca de seguimiento del paquete.
- 3) En la cadena ingress se hace *match* con la primera regla y se activa la traza *nftrace* del paquete, su veredicto en esta cadena es *continue*.
- 4) En prerouting, hace *match* con la regla que comprueba el *state invalid* de los paquetes TCP.

Este paquete TCP tiene activado el flag URG.

Al ser su estado inválido, este paquete no es capaz de crear una entrada en conntrack.

3) TCP RST

```
trace id 8607bf0c netdev ingress ingr rule ip protocol tcp nftrace set 1 counter packets 2 bytes 92 (verdict continue)
trace id 8607bf0c netdev ingress ingr verdict continue
trace id 8607bf0c netdev ingress ingr
trace id 8607bf0c ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 9144 ip length 40 tcp sport 2984 tcp dport 0 tcp flags == rst tcp window 512
trace id 8607bf0c ip filter prerouting rule ip protocol tcp ct state invalid counter packets 0 bytes 0 (verdict continue)
trace id 8607bf0c ip filter prerouting verdict continue
trace id 8607bf0c ip filter prerouting
trace id 8607bf0c ip filter input verdict continue
trace id 8607bf0c ip filter input
```

Figura 6.36 - Camino TCP RST

En la Figura 6.36, se puede ver el camino del paquete TCP RST:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC pasa el paquete a la cadena ingress, la cual activa la marca de seguimiento del paquete.
- 3) En la cadena ingress se hace *match* con la primera regla y se activa la traza *nftrace* del paquete, su veredicto en esta cadena es *continue*.
- 4) En prerouting, hace *match* con la regla que comprueba el *state invalid* de los paquetes TCP.

Este paquete TCP tiene activado el flag RST.

Al ser su estado inválido, este paquete no es capaz de crear una entrada en conntrack.

4) TCP FIN

```
trace id b35d65df netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:le ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 2738 ip length 40 tcp sport 2623 tcp dport 0 tcp flags == fin tcp window 512
trace id b35d65df netdev ingress ingr rule ip protocol tcp nftrace set 1 counter packets 2 bytes 92 (verdict continue)
trace id b35d65df netdev ingress ingr verdict continue
trace id b35d65df netdev ingress ingr
trace id b35d65df ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:le ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 2738 ip length 40 tcp sport 2623 tcp dport 0 tcp flags == fin tcp window 512
trace id b35d65df ip filter prerouting rule ip protocol tcp ct state invalid counter packets 0 bytes 0 (verdict continue)
trace id b35d65df ip filter prerouting verdict continue
trace id b35d65df ip filter prerouting
trace id b35d65df ip filter input verdict continue
trace id b35d65df ip filter input
```

Figura 6.37 - Camino TCP FIN

En la Figura 6.37, se puede ver el camino del paquete TCP FIN:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC pasa el paquete a la cadena ingress, la cual activa la marca de seguimiento del paquete.
- 3) En la cadena ingress se hace *match* con la primera regla y se activa la traza *nftrace* del paquete, su veredicto en esta cadena es *continue*.
- 4) En prerouting, hace *match* con la regla que comprueba el *state invalid* de los paquetes TCP.

Este paquete TCP tiene activado el flag FIN.

Al ser su estado inválido, este paquete no es capaz de crear una entrada en conntrack.

5) TCP ACK

```
root@localhost:/home/dit# nft monitor trace
trace id 9ae1df96 netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:le ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 19918 ip length 40 tcp sport 2353 tcp dport http tcp flags == ack tcp window 512
trace id 9ae1df96 netdev ingress ingr rule ip protocol tcp nftrace set 1 counter packets 5 bytes 300 (verdict continue)
trace id 9ae1df96 netdev ingress ingr verdict continue
trace id 9ae1df96 netdev ingress ingr
trace id 9ae1df96 ip nat pre packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:le ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 19918 ip length 40 tcp sport 2353 tcp dport http tcp flags == ack tcp window 512
trace id 9ae1df96 ip nat pre rule ip protocol tcp counter packets 0 bytes 0 (verdict continue)
trace id 9ae1df96 ip nat pre verdict continue
trace id 9ae1df96 ip nat pre
trace id 9ae1df96 ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:le ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 19918 ip length 40 tcp sport 2353 tcp dport http tcp flags == ack tcp window 512
trace id 9ae1df96 ip filter prerouting rule ip protocol tcp ct state new counter packets 5 bytes 300 continue (verdict continue)
trace id 9ae1df96 ip filter prerouting verdict continue
trace id 9ae1df96 ip filter prerouting
trace id 9ae1df96 ip filter input verdict continue
trace id 9ae1df96 ip filter input
```

Figura 6.38 - Camino TCP ACK

En la Figura 6.38, se puede ver el camino del paquete TCP ACK:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC pasa el paquete a la cadena ingress, la cual activa la marca de seguimiento del paquete.

- 3) En la cadena ingress se hace *match* con la primera regla y se activa la traza *nftrace* del paquete, su veredicto en esta cadena es *continue*.
- 4) En prerouting, hace *match* con la regla que comprueba el *state new* de los paquetes TCP.
Este paquete TCP tiene activado el flag ACK.
Los paquetes ACK tienen una peculiaridad y es que, si son capaces de crear una entrada en conntrack.

6) TCP SYN

```
trace id 4d9395ed netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad eth  
er daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-e  
ct ip ttl 64 ip id 9237 ip length 40 tcp sport rmiregistry tcp dport http tcp flags == syn  
tcp window 512  
trace id 4d9395ed netdev ingress ingr rule ip protocol tcp nftrace set 1 counter packets 6  
bytes 346 (verdict continue)  
trace id 4d9395ed netdev ingress ingr verdict continue  
trace id 4d9395ed netdev ingress ingr  
trace id 4d9395ed ip nat pre packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr  
00:0c:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl  
64 ip id 9237 ip length 40 tcp sport rmiregistry tcp dport http [tcp flags == syn] [tcp windo  
w 512]  
trace id 4d9395ed ip nat pre rule ip protocol tcp counter packets 1 bytes 40 (verdict conti  
nue)  
trace id 4d9395ed ip nat pre verdict continue  
trace id 4d9395ed ip nat pre  
trace id 4d9395ed ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad et  
her daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-  
ect ip ttl 64 ip id 9237 ip length 40 tcp sport rmiregistry tcp dport http tcp flags == syn  
tcp window 512  
trace id 4d9395ed ip filter prerouting rule ip protocol tcp [ct state new] counter packets 6  
bytes 340 continue (verdict continue)  
trace id 4d9395ed ip filter prerouting verdict continue  
trace id 4d9395ed ip filter prerouting  
trace id 4d9395ed ip filter input verdict continue  
trace id 4d9395ed ip filter input
```

Figura 6.39 - Camino TCP SYN

En la Figura 6.39, se puede ver el camino del paquete TCP SYN:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC pasa el paquete a la cadena ingress, la cual activa la marca de seguimiento del paquete.
- 3) En la cadena ingress se hace *match* con la primera regla y se activa la traza *nftrace* del paquete, su veredicto en esta cadena es *continue*.
- 4) En prerouting, hace *match* con la regla que comprueba el *state new* de los paquetes TCP.
Este paquete TCP tiene activado el flag SYN.

Los paquetes SYN, son aquellos que se usan para iniciar una conexión, por lo cual, este tipo de paquetes siempre abren una entrada en conntrack.

7) TCP SYN-ACK

```

trace id 363f0f34 netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 172.168.0.20 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 15197 ip length 40 tcp sport 1454 tcp dport 0 tcp flags == 0x12 tcp window 512
trace id 363f0f34 netdev ingress ingr rule ip saddr 10.10.1.20 nftrace set 1 counter packets 25 bytes 1765 (verdict continue)
trace id 363f0f34 netdev ingress ingr verdict continue
trace id 363f0f34 netdev ingress ingr
trace id 363f0f34 ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 172.168.0.20 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 15197 ip length 40 tcp sport 1454 tcp dport 0 tcp flags == 0x12 tcp window 512
trace id 363f0f34 ip filter prerouting rule ip saddr 10.10.1.20 [ct state invalid] counter packets 0 bytes 0 (verdict continue)
trace id 363f0f34 ip filter prerouting verdict continue
trace id 363f0f34 ip filter prerouting
trace id 363f0f34 ip filter forward verdict continue
trace id 363f0f34 ip filter forward
trace id 363f0f34 ip filter postrouting verdict continue
trace id 363f0f34 ip filter postrouting

```

Figura 6.40 - Camino TCP SYN-ACK

En la Figura 6.37, se puede ver el camino del paquete TCP SYN-ACK:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC pasa el paquete a la cadena ingress, la cual activa la marca de seguimiento del paquete.
- 3) En la cadena ingress se hace *match* con la primera regla y se activa la traza *nftrace* del paquete, su veredicto en esta cadena es *continue*.
- 4) En prerouting, hace *match* con la regla que comprueba el *state invalid* de los paquetes TCP.

Este paquete TCP tiene activado el flag SYN-ACK.

Al ser su estado inválido, este paquete no es capaz de crear una entrada en conntrack.

Esto se debe a que este paquete solo puede ser entendido, en un escenario en el que se hubiera visto un primer paquete de tipo SYN, siendo el SYN-ACK generado como respuesta de este.

7. **[Equipo:Encaminador - Usuario:root]** En Encaminador, se observa el monitor de conntrack para ver las entradas que se han podido crear.

En la Figura 6.41, se puede ver como el paquete TCP SYN enviado por PC1, es capaz de crear una entrada en conntrack al atravesar el *hook input*.

El UPDATE que se puede ver, tras la entrada NEW, es el paquete de respuesta que el servidor envía a PC1.

```

[NEW] tcp      6 120 SYN_SENT src=10.10.1.20 dst=10.10.1.10 sport=1099 dport=80 [UNREPLIED] sr
c=10.10.1.10 dst=10.10.1.20 sport=80 dport=1099
[UPDATE] tcp      6 60 SYN_RECV src=10.10.1.20 dst=10.10.1.10 sport=1099 dport=80 src=10.10.1.10
dst=10.10.1.20 sport=80 dport=1099

```

Figura 6.41 - Entrada conntrack TCP-SYN

En la Figura 6.42, se puede ver como el paquete TCP ACK enviado por PC1, es capaz de crear una entrada en conntrack al atravesar el *hook input*.

Pero al detectar el servidor que esta entrada no es propia, es decir, que no se había establecido conexión previa con PC1, automáticamente la elimina sin esperar a que llegue el final de su *timeout*.

```

root@localhost:/home/dit# conntrack -E --event-mask ALL
[NEW] tcp      6 300 ESTABLISHED src=10.10.1.20 dst=10.10.1.10 sport=2353 dport=80 [UNREPLIED]
src=10.10.1.10 dst=10.10.1.20 sport=80 dport=2353
[DESTROY] tcp      6 src=10.10.1.20 dst=10.10.1.10 sport=2353 dport=80 [UNREPLIED] src=10.10.1.10
dst=10.10.1.20 sport=80 dport=2353
^Cconntrack v1.4.6 (conntrack-tools) - 2 flow events have been shown

```

Figura 6.42 - Entrada conntrack TCP-ACK

Conclusión del ejemplo:

Como conclusión de este punto, se presenta un cuadro resumen (Tabla 15), con los distintos tipos de paquetes TCP que podremos encontrarnos, así como información de si son capaces de abrir una entrada en conntrack y si su estado es *invalid*.

Paquete TCP	Puede abrir entrada en conntrack	Estado invalid
SYN	SI	NO
ACK	SI	NO
PUSH	NO	SI
RST	NO	SI
URG	NO	SI
RST	NO	SI
SYN-ACK	NO	SI

Tabla 15 - Resumen paquetes TCP invalid

A continuación, se pasará a ver el protocolo UDP en conntrack.

6.4 UDP Conntrack

UDP es un protocolo que no está orientado a la conexión, por lo que el lector puede preguntarse, cómo es posible que el módulo conntrack, pueda realizar el seguimiento de la conexión de un protocolo, que es no orientado a conexión.

Se recomienda la revisión del anexo de protocolos, en especial, el punto “Protocolo UDP (User Datagram Protocol)”[53].

Al comienzo de este capítulo, se vio que el módulo conntrack realiza el seguimiento de los flujos de comunicación, por lo que UDP, al igual que TCP, también establece flujos de comunicación.

La diferencia con el protocolo visto anteriormente, es que UDP no comienza con una fase de establecimiento de la conexión, sino que directamente envía sus datos y obtiene sus respuestas.

En este protocolo, sí se puede decir que el primer paquete de un nuevo flujo de comunicación que sea visto por el módulo conntrack, crea una entrada con la categoría NEW, mientras que el segundo de ellos completará el flujo y actualizará su estado.

6.4.1 Ejemplo UDP conntrack.

Descripción del Ejemplo:

Para realizar el ejemplo con UDP, se partirá de la estructura creada en “TCP Conntrack”, la cual se basa en el “Anexo E - Escenario 2 Equipos”.

En este ejemplo;

- PC1 generará consultas al DNS mediante el uso del comando nslookup
- Encaminador será el encargado de aplicar las reglas de nftables, así como de observar las entradas del módulo conntrack y de monitorizar el paso del paquete por las diferentes reglas.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** A las reglas vistas anteriormente, se le añade la siguiente regla en la cadena post de la tabla nat, puesto que, al salir a una red distinta, necesitamos aplicarle *masquerade* al paquete que se va a encaminar.

Equipo:Encaminador Usuario:root

```
$ nft add rule ip nat post ip saddr 10.10.1.20 counter masquerade
```

2. **[Equipo:Encaminador - Usuario:root]** Se abren dos terminales en el equipo encaminador, además se abre Wireshark y se pone a escuchar en la interfaz “ens38”.
 - Terminal 1: Se abre conntrack en modo monitor de eventos, usando la opción de mostrar todos los eventos.
 - Terminal 2: Se abre un monitor de reglas nftables.

Equipo:Encaminador Usuario:root

```
#Terminal 1  
$ conntrack -E --event-mask ALL  
  
#Terminal 2  
$ nft monitor trace
```

3. **[Equipo:PC1 - Usuario:root]** Se usa el comando nslookup desde PC1, para pedir la dirección de la web trajano.

Equipo:PC1 Usuario:root

```
$ nslookup trajano.us.es
```

4. **[Equipo:Encaminador - Usuario:root]** Se observa en el monitor de reglas, el camino del paquete de petición al servidor DNS 8.8.8.8 y la captura de Wireshark realizada.

```

trace id 46292878 netdev filteringress ingress packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether dad
dr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 8.8.8.8 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 3949
ip length 59 udp sport 52777 udp dport domain udp length 39
trace id 46292878 netdev filteringress ingress rule nftrace set 1 (verdict continue)
trace id 46292878 netdev filteringress ingress verdict continue
trace id 46292878 netdev filteringress ingress
trace id 46292878 ip filter prerouting verdict continue
trace id 46292878 ip filter prerouting
trace id 46292878 ip nat prerouting verdict continue
trace id 46292878 ip nat prerouting
trace id 46292878 ip filter forward verdict continue
trace id 46292878 ip filter forward
trace id 46292878 inet filter forward verdict continue
trace id 46292878 inet filter forward
trace id 46292878 ip filter postrouting verdict continue
trace id 46292878 ip filter postrouting
trace id 46292878 ip nat postrouting packet: oif "ens33" ip saddr 10.10.1.20 ip daddr 8.8.8.8 ip dscp cs0
ip ecn not-ect ip ttl 63 ip id 3949 ip length 59 udp sport 52777 udp dport domain udp length 39
trace id 46292878 ip nat postrouting rule ip saddr 10.10.1.20 counter packets 63 bytes 4436 masquerade (verdict accept)

```

Figura 6.43 - Camino de un paquete UDP

En la Figura 6.43 se puede ver el camino del paquete UDP enviado por PC1:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC pasa el paquete a la cadena filteringress, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en filteringress es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 4) En prerouting se procesa el paquete y en una fase temprana se clasifica con *state NEW*, su veredicto es *continue*, por lo que continúa con su procesado.
- 5) Se consulta el encaminamiento del paquete, determinando que este ha de ser encaminado.
- 6) Al tener que ser encaminado, el paquete continúa su camino por el *hook forward*, siendo aquí su veredicto *continue*.
- 7) El paquete llega al *hook postrouting*, donde antes de abandonar el equipo, se le aplica la regla de NAT y se realiza el *masquerade*, cambiando su Ip origen por la de la interfaz “ens33” del equipo Encaminador.

En este punto se crea también la entrada de conntrack con estado NEW, en la cual se recoge los datos del paquete UDP.

En la Figura 6.44 se puede ver, como se realiza la petición de IP al servidor DNS, de forma que la petición del paquete entra por la interfaz “ens38” y sale por la interfaz “ens33” después de haberle realizado el *masquerade* para que pueda salir al exterior.

	7 7.528332127	10.10.1.20	8.8.8.8	DNS	73 Standard query 0x2f0b A trajano.us.es
	8 7.589609151	8.8.8.8	10.10.1.20	DNS	89 Standard query response 0x2f0b A trajano.us.es A 193.147.162.139
► Frame 7: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0					
► Ethernet II, Src: Vmware_62:ed:ad (00:0c:29:62:ed:ad), Dst: Vmware_f0:01:1e (00:0c:29:f0:01:1e)					
► Internet Protocol Version 4, Src: 10.10.1.20, Dst: 8.8.8.8					
► User Datagram Protocol, Src Port: 55072, Dst Port: 53					
Source Port: 55072					
Destination Port: 53					
Length: 39					
Checksum: 0xad7f [unverified]					
[Checksum Status: Unverified]					
[Stream index: 1]					
► Domain Name System (query)					

Figura 6.44 - Captura de paquete UDP

5. [Equipo:PC1 - Usuario:root] En el equipo PC1, se observa la salida del comando nslookup y se comprueba el éxito de la petición DNS.

La respuesta del servidor DNS viene de vuelta y llega al equipo PC1, indicándole la Ip del servidor de trajano.us.es, tal como se puede ver en la Figura 6.45

```
[root@localhost dit]# nslookup trajano.us.es
Server:      8.8.8
Address:     8.8.8.8#53

Non-authoritative answer:
Name:   trajano.us.es
Address: 193.147.162.130

[root@localhost dit]#
```

Figura 6.45 - Comando nslookup para consultar DNS

6. [Equipo:Encaminador - Usuario:root] En el equipo Encaminador, se revisan las entradas que se han ido creando en conntrack.

En la Figura 6.46 podemos ver, que ante el paquete UDP de petición al servidor DNS, se crea en conntrack, una entrada con la categoría NEW y con el flag UNREPLIED activado.

Al recibir la respuesta y que esta cruce por el *hook postrouting*, se actualiza la entrada anterior y se elimina el flag que indicaba que el flujo de datos no había visto respuesta.

En la Figura 6.46 también se puede ver la entrada que imprime conntrack al consultar su listado y podemos ver el valor de los campos indicados en el primer punto de este capítulo.

```
root@localhost:/home/dit# conntrack -L
udp      17 22 src=10.10.1.20 dst=8.8.8.8 sport=44375 dport=53 src=8.8.8.8 dst=192.168.106.137 sport=53 dport=44375 mark=0 use=1
[NEW] udp      17 30 src=10.10.1.20 dst=8.8.8.8 sport=55072 dport=53 [UNREPLIED] src=8.8.8.8 dst=192.168.106.137 sport=53 dport=55072
[UPDATE] udp      17 30 src=10.10.1.20 dst=8.8.8.8 sport=55072 dport=53 src=8.8.8.8 dst=192.168.106.137 sport=53 dport=55072
[NEW] udp      17 30 src=10.10.1.20 dst=8.8.8.8 sport=52777 dport=53 [UNREPLIED] src=8.8.8.8 dst=192.168.106.137 sport=53 dport=52777
[UPDATE] udp      17 30 src=10.10.1.20 dst=8.8.8.8 sport=52777 dport=53 src=8.8.8.8 dst=192.168.106.137 sport=53 dport=52777
```

Figura 6.46 - Entradas conntrack paquete UDP

En la Figura 6.47, se puede observar un resumen del estado que asigna conntrack al paquete UDP conforme se produce la comunicación con el servidor DNS.

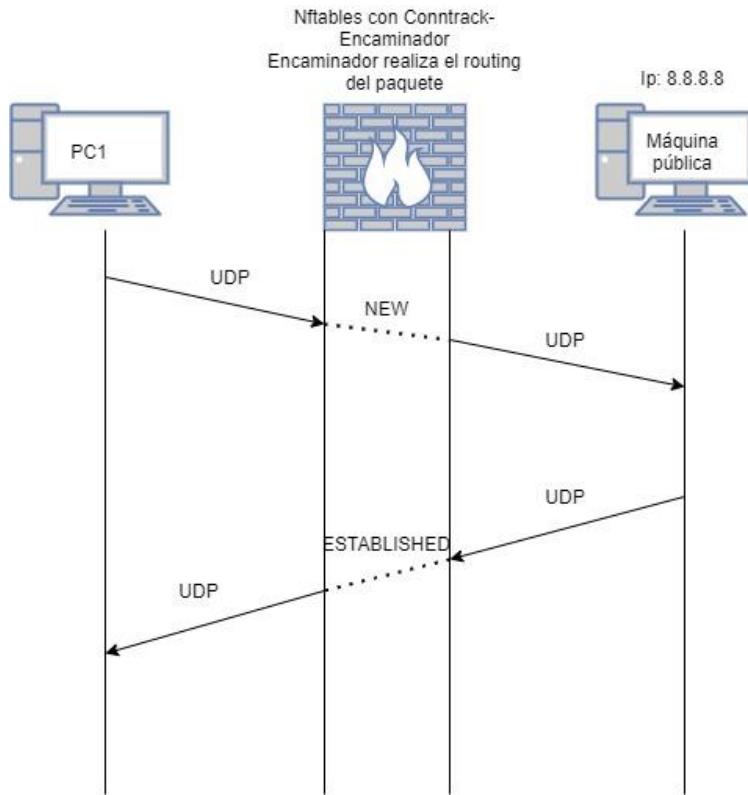


Figura 6.47 - Esquema estados conntrack UDP

En la Figura 6.48 se puede observar el camino que hace cada uno de los dos paquetes de UDP que ve Encaminador (pregunta y respuesta).

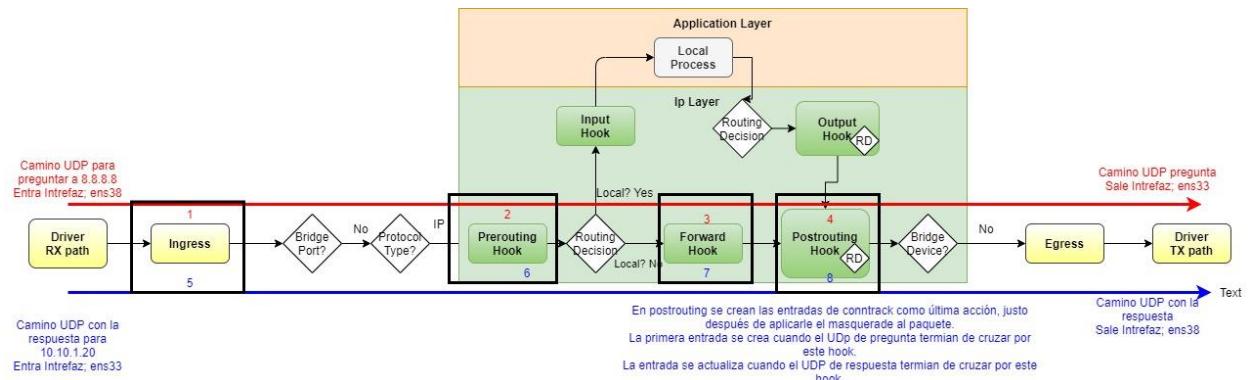


Figura 6.48 - Camino paquetes conntrack UDP

Conclusión del ejemplo:

De este ejemplo, cabe destacar, el estudio de la Figura 6.47 y Figura 6.48, ya que en ellas reside la base conceptual del ejemplo, mostrando los estados que obtienen los paquetes UDP durante su transmisión, así como visualizando el camino que estos siguen a través de los *hooks* de Netfilter.

A diferencia de TCP, con este protocolo, no tenemos paquetes que no abran o no puedan ser asociados a una entrada conntrack.

6.5 ICMP Conntrack

Con el protocolo ICMP, ocurre lo mismo que con el protocolo UDP visto anteriormente, ambos, son protocolos que no están orientados a la conexión, pero como ya se comentó anteriormente, esto no supone ningún impedimento para poder realizar el seguimiento de su flujo de comunicación.

Como también se dijo anteriormente, no todos los tipos de paquetes son capaces de crear una entrada de conntrack, de forma que si el primer paquete que esta ve, es de tipo echo request, creará la entrada con la categoría NEW, mientras que si el paquete es de tipo echo-reply y conntrack no tiene ninguna entrada anterior con la que poder asociarlo, lo marcará como *invalid*.

Se recomienda la revisión del anexo de protocolos, en especial, el punto “Protocolo ICMP (Internet Control Message Protocol)” [54].

6.5.1 Ejemplo ICMP conntrack.

Descripción del ejemplo:

Para realizar los ejemplos de este punto, se usará el escenario de tres equipos “Anexo F - Escenario 3 Equipos”.

- PC1 será el encargado de generar el tráfico ICMP, bien con el comando ping o usando hping3.
- Encaminador será el encargado de usar nftables y de visualizar las entradas de conntrack, así como monitorizar el camino del paquete a través de la estructura de tablas y cadenas de Netfilter.
- PC2 será el encargado de recibir ciertas peticiones y de probar el caso del *reject*.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Para el ejemplo de ICMP, se crea una tabla filter, una tabla nat y otra netdev.

```
Equipo: Encaminador Usuario: root

$ nft add table ip nat
$ nft add table ip filter
$ nft add table netdev filteringress
```

2. **[Equipo:Encaminador - Usuario:root]** Se crean los siguientes grupos de cadenas:

- 1) **Tabla filter**: cadenas prerouting, input, forward, postrouting y output.
- 2) **Tabla nat**: cadenas prerouting y postrouting.
- 3) **Tabla netdev**: cadena filteringress.

```
Equipo: Encaminador Usuario: root

$ nft add chain 'ip nat prerouting { type nat hook prerouting priority -1 ; }'
$ nft add chain 'ip nat postrouting { type nat hook postrouting priority -1 ; }'
$ nft add chain 'ip filter prerouting { type filter hook prerouting priority -1 ; }'
$ nft add chain 'ip filter postrouting { type filter hook postrouting priority -1 ; }'
```

```

$ nft add chain 'ip filter input { type filter hook input priority -1 ; }'
$ nft add chain 'ip filter forward { type filter hook forward priority 0 ; }'
$ nft add chain 'ip filter output { type filter hook output priority -1 ; }'
$ nft add chain 'ip filter prerouting { type filter hook prerouting priority -1 ; }'
$ nft add 'chain netdev filteringress ingress { type filter hook ingress device ens38 priority 0 ; }'

```

3. [Equipo:Encaminador - Usuario:root] Se crean las siguientes reglas:

- 1) En la cadena filteringress, se añade una regla que active *nftrace* para todo el tráfico entrante.
- 2) En la cadena prerouting de filter, se añade una regla que contabilice el tráfico ICMP cuyo *state* sea *invalid*.
- 3) En la cadena postrouting de nat, se añade una regla para contabilizar y realizar el *masquerade* al tráfico cuya Ip origen sea la de PC1.

Equipo: Encaminador **Usuario:**root

```

$ nft add rule netdev filteringress nftrace set 1
$ nft add rule ip filter prerouting ip protocol tcp ct state invalid counter continue
$ nft add rule ip nat postrouting ip saddr 10.10.1.20 counter masquerade

```

4. [Equipo:Encaminador - Usuario:root] Se abren dos terminales en el equipo Encaminador:

- Terminal 1: Se abre conntrack en modo monitor de eventos, usando la opción de mostrar todos los eventos.
- Terminal 2: Se abre un monitor de reglas nftables.

Equipo:Encaminador **Usuario:**root

```

#Terminal 1
$ conntrack -E --event-mask ALL

#Terminal 2
$ nft monitor trace

```

5. [Equipo:PC1 - Usuario:root] Se abre Wireshark y se pone a escuchar en la interfaz “ens37”.

6. [Equipo:PC1 - Usuario:root] Se lanza un ICMP echo-request desde PC1 hacia la dirección de PC2.

Equipo: PC1 **Usuario:** root

```
$ ping -c 1 172.168.0.20
```

7. [Equipo:PC1 - Usuario:root] Se comprueba la salida de Wireshark.

Como se puede comprobar en las Figura 6.49 y Figura 6.50, la petición de echo request llega a Encaminador y este la encamina hacia PC2 saliendo el paquete por la interfaz “ens39”.

Al recibir el echo request, PC2 responde con el echo-reply y al llegar a Encaminador, este lo encamina hacia PC1, el cual recibe la respuesta y da por completado el ping.

No.	Time	Source	Destination	Protocol	Length:Info
1	0.000000000	10.10.1.20	172.168.0.20	ICMP	98 Echo (ping) request id=0x3061, seq=1/256, ttl=64 (reply in 2)
2	0.055449481	172.168.0.20	10.10.1.20	ICMP	98 Echo (ping) reply id=0x3061, seq=1/256, ttl=63 (request in 1)

```

▶ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▶ Ethernet II, Src: VMware_62:ed:ad (00:0c:29:62:ed:ad), Dst: VMware_f0:01:1e (00:0c:29:f0:01:1e)
▶ Internet Protocol Version 4, Src: 10.10.1.20, Dst: 172.168.0.20
▼ Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0xd0ef [correct]
  [Checksum Status: Good]
  Identifier (BE): 12385 (0x3061)
  Identifier (LE): 24880 (0x6130)
  Sequence number (BE): 1 (0x0001)
  Sequence number (LE): 256 (0x0100)
  [Response frame: 2]
  Timestamp from icmp data: Feb 2, 2021 19:22:55.000000000 CET
  [Timestamp from icmp data (relative): 7199.496886467 seconds]
  Data (48 bytes)

```

Figura 6.49 - Captura echo-request y echo-reply ICMP conntrack

8. [Equipo:Encaminador - Usuario:root] Se comprueba la salida del monitor de nftrace.

Esta salida puede verse en la Figura 6.50.

```

root@localhost:/home/dit# nft monitor trace
trace id b2f7cb3c netdev filteringress ingress packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether dad
dr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 172.168.0.20 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id
45278 ip length 84 icmp type echo-request icmp code 0 icmp id 11992 icmp sequence 1
trace id b2f7cb3c netdev filteringress ingress rule nftrace set 1 (verdict continue)
trace id b2f7cb3c netdev filteringress ingress verdict continue
trace id b2f7cb3c netdev filteringress ingress
trace id b2f7cb3c ip filter prerouting verdict continue
trace id b2f7cb3c ip filter prerouting
trace id b2f7cb3c ip nat prerouting verdict continue
trace id b2f7cb3c ip nat prerouting
trace id b2f7cb3c ip filter forward verdict continue
trace id b2f7cb3c ip filter forward
trace id b2f7cb3c inet filter forward verdict continue
trace id b2f7cb3c inet filter forward
trace id b2f7cb3c ip filter postrouting verdict continue
trace id b2f7cb3c ip filter postrouting
trace id b2f7cb3c ip nat postrouting packet: oif "ens39" ip saddr 10.10.1.20 ip daddr 172.168.0.20 ip dscp
cs0 ip ecn not-ect ip ttl 63 ip id 45278 ip length 84 icmp type echo-request icmp code 0 icmp id 11992 ic
mp sequence 1
trace id b2f7cb3c ip nat postrouting rule ip saddr 10.10.1.20 counter packets 111 bytes 7783 masquerade (v
erdict accept)

```

Figura 6.50 - Camino paquete echo-request

En la Figura 6.50, se puede ver el camino del paquete ICMP enviado por PC1 hacia PC2:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC pasa el paquete a la cadena filteringress, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en filteringress es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 4) En prerouting, se procesa y en una fase temprana se clasifica el paquete con *state NEW*, su veredicto es *continue*, por lo que se continua con su procesado.
- 5) Se consulta el encaminamiento del paquete, determinando que este ha de ser encaminado hacia PC2.
- 6) Al tener que ser encaminado, el paquete continúa su camino por el *hook forward*, siendo aquí su veredicto *continue*.
- 7) El paquete llega al *hook postrouting*, donde, antes de abandonar el equipo se le aplica la regla de NAT y se realiza el *masquerade*, cambiando su Ip origen por la de la interfaz “ens39” del

equipo Encaminador.

En este punto, se crea también la entrada de conntrack con estado NEW, en la cual, se recogen los datos del paquete ICMP.

El paquete echo-reply, realiza el mismo camino que el realizado por el request, categorizando este paquete como ESTABLISHED en el *hook prerouting* y actualizando la entrada de conntrack cuando este cruza por el *hook postrouting*, camino a PC1.

9. [Equipo:Encaminador - Usuario:root] Se comprueba la salida del monitor de conntrack.

Esta salida puede verse en la Figura 6.51.

```
root@localhost:/home/dit# conntrack -E --event-mask ALL
[NEW] icmp    1 30 src=10.10.1.20 dst=172.168.0.20 type=8 code=0 id=11992 [UNREPLIED] src=172
.168.0.20 dst=172.168.0.10 type=0 code=0 id=11992
[UPDATE] icmp    1 30 src=10.10.1.20 dst=172.168.0.20 type=8 code=0 id=11992 src=172.168.0.20 ds
t=172.168.0.10 type=0 code=0 id=11992
^Cconntrack v1.4.6 (conntrack-tools): 2 flow events have been shown.
root@localhost:/home/dit#
```

Figura 6.51 - Entradas conntrack creadas por comunicación ICMP

La primera entrada, la cual podemos ver con el estado NEW y el flag UNREPLIED activado, se crea cuando el paquete icmp-request, cruza el *hook postrouting* de Encaminador , dirigiéndose hacia PC2.

La actualización de la entrada, tiene lugar cuando el paquete icmp-reply, cruza el *hook postrouting* de Encaminador, dirigiéndose así hacia PC1, en este momento también se pone a 0 el flag de UNREPLIED

10. [Equipo:PC1 - Usuario:root] Ahora se comprobará las salidas del monitor de nftrace y de conntrack, enviando un icmp-request hacia PC2, en lugar de un icmp-reply.

Equipo: PC1 **Usuario:** root

```
#-C indica el tipo de ICMP y -K el código ICMP
$ hping3 -1 -C 0 -K 0 172.168.0.20
```

11. [Equipo:Encaminador - Usuario:root] Se comprueba la salida del monitor de nftrace y de conntrack.

Observamos, que no se crea la entrada en conntrack y que al ser este un paquete de respuesta y no tener flujo asociado ni poder crearlo, su estado es el de inválido, tal y como se puede comprobar en la Figura 6.52.

En la Figura 6.52 se puede ver el camino del paquete ICMP reply enviado por PC1 hacia PC2:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena filteringress, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en filteringress es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 4) En prerouting se procesa y en una fase temprana, se clasifica el paquete con *state INVALID*, ya que no puede crear una entrada dentro de conntrack, ni tampoco existe entrada a la que poder asociar ese paquete.
- 5) Se consulta el encaminamiento del paquete, determinando que este ha de ser encaminado hacia PC2.

- 6) Al tener que ser encaminado, el paquete continua su camino por el *hook forward*, siendo aquí su veredicto *continue*.
- 7) El paquete llega al *hook postrouting*, al ser INVALID, este no entra en la cadena NAT, por lo que al no realizársele el *masquerade*, la respuesta nunca llega a PC1.

The screenshot shows two terminal windows side-by-side. The left window displays the output of the command `nft monitor trace`. It shows a sequence of traces for an ICMP echo-request packet. The packet enters via interface `ens38` (source IP `10.10.1.20`) and is processed by several Netfilter hooks: `ingress`, `nftrace set 1` (verdict `continue`), `netdev filteringress` (verdict `continue`), `ip filter prerouting` (verdict `continue`), `inet filter forward` (verdict `continue`), and `ip filter postrouting` (verdict `continue`). The packet then reaches the destination via interface `eth0` (destination IP `172.168.0.20`). The right window shows the output of the command `conntrack -E --event-mask ALL`, which lists the connection entries in the conntrack table. One entry is highlighted with a red box, showing the `state` as `invalid`.

```

root@localhost:/home/dit# nft monitor trace
trace id 74ff362a netdev filteringress ingress packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:le ip saddr 10.10.1.20 ip daddr 172.168.0.20 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 6770 ip length 28 icmp type echo-reply icmp code 0 icmp id 8497 icmp sequence 0
trace id 74ff362a netdev filteringress ingress rule nftrace set 1 (verdict continue)
trace id 74ff362a netdev filteringress ingress verdict continue
trace id 74ff362a netdev filteringress ingress
trace id 74ff362a ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:le ip saddr 10.10.1.20 ip daddr 172.168.0.20 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 6770 ip length 28 icmp type echo-reply icmp code 0 icmp id 8497 icmp sequence 0
trace id 74ff362a ip filter prerouting rule ip protocol icmp ct state invalid counter packets 0 bytes 0 continue (verdict continue)
trace id 74ff362a ip filter prerouting verdict continue
trace id 74ff362a ip filter prerouting
trace id 74ff362a ip filter forward verdict continue
trace id 74ff362a ip filter forward
trace id 74ff362a inet filter forward verdict continue
trace id 74ff362a inet filter forward
trace id 74ff362a ip filter postrouting verdict continue
trace id 74ff362a ip filter postrouting
trace id 33f74b11 netdev filteringress ingress packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:le arp operation request
trace id 33f74b11 netdev filteringress ingress rule nftrace set 1 (verdict continue)
trace id 33f74b11 netdev filteringress ingress verdict continue
trace id 33f74b11 netdev filteringress ingress
root@localhost:/home/dit# conntrack -E --event-mask ALL

```

Figura 6.52 - Envío de paquete echo-reply

Conclusión del ejemplo:

Este ejemplo, sirve para comprobar, que la afirmación que se puede observar en numerosas guías y sitios web, la cual, asegura que las respuestas pueden crear una entrada en conntrack, es falsa. Por ello, es importante que un administrador de red tenga en cuenta, que estos paquetes necesitan estar asociados a un flujo con entrada en conntrack, en caso contrario, el estado de los mismos será *invalid*.

También sirve para comprobar el camino que realizan los paquetes ICMP tanto de petición como de respuesta a lo largo del sistema de Netfilter.

6.5.2 Ejemplo ICMP conntrack reject.

Descripción del ejemplo:

Para el siguiente ejemplo, se creará una regla que realice el *reject* del tráfico ICMP para la máquina local de PC2, de forma que se podrá ver como se crea la entrada de conntrack, así como el *state related* del paquete generado por ICMP *reject*, viendo como se considera, que pertenece al flujo creado por el echo-request de PC1.

Para este ejemplo, se usará el escenario de 3 equipos “Anexo F - Escenario 3 Equipos” y se partirá de las reglas creadas en el “Ejemplo ICMP conntrack.”.

Análisis del ejemplo:

1. [Equipo:PC2 - Usuario:root] Se debe crear una estructura de nftables en PC2 y añadir la siguientes reglas.
 - Una tabla filter con las cadenas de prerouting e input.
 - Añadir tres reglas en la cadena input, una por cada tipo de tráfico que realice su *reject*.

```
Equipo:PC2 Usuario:root

#Creación de tabla y cadenas
$ nft add table ip filter
$ nft add chain ip' filter pre { type filter hook prerouting priority 0 ; }
$ nft add chain ip' filter input { type filter hook input priority 0 ; }

#Creación de reglas
$ nft add rule ip filter input ip protocol udp reject with icmp type net-unreachable
$ nft add rule ip filter input ip protocol tcp reject with icmp type net-unreachable
$ nft add rule ip filter input ip protocol icmp reject with icmp type net-unreachable
```

2. [Equipo:PC2 - Usuario:root] Se comprueba que la estructura de reglas de PC2 se corresponde con la vista en la Figura 6.53.

```
Equipo:PC2 Usuario:root

$ nft list ruleset
```

```
[root@localhost dit]# nft list ruleset
table ip filter {
    chain pre {
        type filter hook prerouting priority filter; policy accept;
    }

    chain input {
        type filter hook input priority filter; policy accept;
        ip protocol tcp reject with icmp type net-unreachable
        ip protocol udp reject with icmp type net-unreachable
        ip protocol icmp reject with icmp type net-unreachable
    }
}
[root@localhost dit]#
```

Figura 6.53 - Reglas reject de ICMP

3. [Equipo:Encaminador - Usuario:root] En Encaminador, se añade la siguiente regla para poder ver si el paquete tiene el estado de *related*, de forma que la estructura de nftables queda como se puede ver en la Figura 6.54

```
Equipo:Encaminador Usuario:root

$ nft add rule ip filter prerouting ip protocol icmp ct state related counter continue
```

```
#Se muestra el ruleset de nftables
$ nft list ruleset
```

```
table ip filter {
    chain prerouting {
        type filter hook prerouting priority -1; policy accept;
        ip protocol icmp ct [state related] counter packets 0 bytes 0 continue
        ip protocol tcp ct [state related] counter packets 0 bytes 0 continue
        ip protocol icmp ct state invalid counter packets 1 bytes 28 continue
        ip protocol tcp ct state invalid counter packets 5 bytes 200 continue
        ip protocol tcp ct state { established, new } counter packets 12110 bytes 107926659 continue
    }

    chain postrouting {
        type filter hook postrouting priority -1; policy accept;
    }

    chain output {
        type filter hook output priority -1; policy accept;
        nftrace set 1
        ip protocol icmp ct state invalid counter packets 0 bytes 0 continue
        ip protocol icmp ct state related counter packets 6 bytes 504 continue
    }

    chain input {
        type filter hook input priority -1; policy accept;
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
    }
}
```

Figura 6.54 - Paquete reject con estado related

4. **[Equipo:Encaminador - Usuario:root]** Se abren dos terminales en el equipo Encaminador:

- Terminal 1: Se abre conntrack en modo monitor de eventos, usando la opción de mostrar todos los eventos.
- Terminal 2: Se abre un monitor de reglas nftables.

```
Equipo:Encaminador Usuario:root

#Terminal 1
$ conntrack -E --event-mask ALL

#Terminal 2
$ nft monitor trace
```

5. **[Equipo:PC1- Usuario:root]** Se realiza el ping desde PC1 con destino PC2.

```
Equipo:PC1 Usuario:root

$ ping -c 2 172.168.0.2
```

```
[root@localhost dit]# ping -c 2 172.168.0.20
PING 172.168.0.20 (172.168.0.20) 56(84) bytes of data.
From 172.168.0.20 icmp_seq=1 Destination Net Unreachable
From 172.168.0.20 icmp_seq=2 Destination Net Unreachable
--- 172.168.0.20 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 25ms
```

Figura 6.55 - Respuesta de la regla reject

En las Figura 6.55, se puede ver que se envía el ping desde PC1 y llega la respuesta del *reject* desde PC2.

6. [Equipo:Encaminador - Usuario:root] Se comprueba la salida del monitor de nftrace.

Esta salida puede verse en la Figura 6.56 y en la Figura 6.57, siendo la primera el camino del paquete echo-request y la segunda, el camino del paquete producido por el *reject* de PC2.

```
trace id 1412a55b netdev filteringress ingress packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 172.168.0.20 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 27188 ip length 84 icmp type echo-request icmp code 0 icmp id 13430 icmp sequence 1
trace id 1412a55b netdev filteringress ingress rule nftrace set 1 (verdict continue)
trace id 1412a55b netdev filteringress ingress verdict continue
trace id 1412a55b netdev filteringress ingress
trace id 1412a55b ip filter prerouting verdict continue
trace id 1412a55b ip filter prerouting
trace id 1412a55b ip nat prerouting verdict continue
trace id 1412a55b ip nat prerouting
trace id 1412a55b ip filter forward verdict continue
trace id 1412a55b ip filter forward
trace id 1412a55b inet filter forward verdict continue
trace id 1412a55b inet filter forward
trace id 1412a55b ip filter postrouting verdict continue
trace id 1412a55b ip filter postrouting
trace id 1412a55b ip nat postrouting packet: oif "ens39" ip saddr 10.10.1.20 ip daddr 172.168.0.20 ip dscp cs0 ip ecn not-ect ip ttl 63 ip id 27188 ip length 84 icmp type echo-request icmp code 0 icmp id 13430 icmp sequence 1
trace id 1412a55b ip nat postrouting rule ip saddr 10.10.1.20 counter packets 126 bytes 8947 masquerade (verdict accept)
```

Figura 6.56 - Camino paquete echo-request ejemplo reject

```
trace id 290d7293 netdev filterens39 ingress packet: iif "ens39" ether saddr 00:0c:29:91:4b:f2 ether daddr 00:0c:29:f0:01:28 ip saddr 172.168.0.20 ip daddr 172.168.0.10 ip dscp cs6 ip ecn not-ect ip ttl 64 ip id 8501 ip length 112 icmp type destination-unreachable icmp code 0 icmp id 0 icmp sequence 0
trace id 290d7293 netdev filterens39 ingress rule nftrace set 1 (verdict continue)
trace id 290d7293 netdev filterens39 ingress verdict continue
trace id 290d7293 netdev filterens39 ingress
trace id 290d7293 ip filter prerouting packet: iif "ens39" ether saddr 00:0c:29:91:4b:f2 ether daddr 00:0c:29:f0:01:28 ip saddr 172.168.0.20 ip daddr 172.168.0.10 ip dscp cs6 ip ecn not-ect ip ttl 64 ip id 8501 ip length 112 icmp type destination-unreachable icmp code 0 icmp id 0 icmp sequence 0
trace id 290d7293 ip filter prerouting rule ip protocol icmp ct state related counter packets 1 bytes 112 continue (verdict continue)
trace id 290d7293 ip filter prerouting verdict continue
trace id 290d7293 ip filter prerouting
trace id 290d7293 ip filter forward verdict continue
trace id 290d7293 ip filter forward
trace id 290d7293 inet filter forward verdict continue
trace id 290d7293 inet filter forward
trace id 290d7293 ip filter postrouting verdict continue
trace id 290d7293 ip filter postrouting
```

Figura 6.57 - Camino paquete echo-reply generado por reject

En la Figura 6.56 se puede ver el camino del paquete ICMP request enviado por PC1 hacia PC2:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena filteringress, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en filteringress es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 4) En prerouting se procesa y en una fase temprana, se clasifica el paquete con *state NEW*, su veredicto es *continue*, por lo que se continúa con su procesado.
- 5) Se consulta el encaminamiento del paquete, determinando que este ha de ser encaminado hacia PC2.
- 6) Al tener que ser encaminado, el paquete continua su camino por el *hook forward*, siendo aquí su veredicto de *continue*.
- 7) El paquete llega al *hook postrouting*, donde antes de abandonar el equipo, se le aplica la regla

de NAT y se le realiza *masquerade* cambiando su Ip origen por la de la interfaz “ens39” del equipo Encaminador.

En la Figura 6.57 se puede ver el camino del paquete ICMP reply enviado por PC2 hacia PC1:

- 1) El paquete sale de PC2 y llega a Encaminador por la interfaz “ens39”
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena filteringress, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en filteringress es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 4) En prerouting se procesa y en una fase temprana se clasifica el paquete con *state RELATED*, ya que este paquete está relacionado con el flujo de comunicación que creó el paquete ICMP-request.

Podemos ver como hace *match* con la regla *related* de ICMP y aumenta su contador, su veredicto es *continue*.

- 5) Se consulta el encaminamiento del paquete, determinando que este ha de ser encaminado hacia PC1.
 - 6) Al tener que ser encaminado, el paquete continúa su camino por el *hook forward*, siendo aquí su veredicto de *continue*.
 - 7) El paquete llega al *hook postrouting*, pero como no es la respuesta que la entrada de conntrack esperaba, no la actualiza y al ser este un paquete de respuesta, tampoco es capaz de crear una entrada, por lo que la entrada existente se quedará con la bandera de UNREPLIED.
7. [Equipo:Encaminador - Usuario:root] Se comprueba la salida del monitor de conntrack y del *ruleset* de nftables.
- En la Figura 6.58, se puede ver que en el monitor de eventos de conntrack, solo se produce un evento y es la creación de una nueva entrada en conntrack, la cual tiene activa el flag de UNREPLIED ya que no es capaz de ver respuesta a ese flujo de tráfico.
 - La Figura 6.59, nos muestra la entrada de conntrack del paquete ICMP, la cual nunca ve la respuesta al paquete de echo request, ya que el paquete producido por el *reject* es RELATED a ese flujo, pero no pertenece como tal a este.

Es por ello que, pasados los 30 segundos de tiempo establecido para este tipo de entradas, se borra.

- Además de ello, se puede ver en la Figura 6.60, como se ha incrementado el contador de paquete ICMP de *state related*.

```
root@localhost:/home/dit# conntrack -E --event-mask ALL
[NEW] icmp      1 30 src=10.10.1.20 dst=172.168.0.20 type=8 code=0 id=13430 [UNREPLIED] src=172.168.0.20 dst=172.168.0.10 type=0 code=0 id=13430
^Cconntrack v1.4.6 (conntrack-tools): 1 flow events have been shown.
```

Figura 6.58 - Evento conntrack del request del ejemplo reject

```
root@localhost:/home/dit# conntrack -L
icmp      1 9 src=10.10.1.20 dst=172.168.0.20 type=8 code=0 id=13430 [UNREPLIED]
src=172.168.0.20 dst=172.168.0.10 type=0 code=0 id=13430 mark=0 use=1
conntrack v1.4.6 (conntrack-tools): 1 flow entries have been shown.
root@localhost:/home/dit#
```

Figura 6.59 - Entrada conntrack del request del ejemplo reject

```

table ip filter {
    chain prerouting {
        type filter hook prerouting priority -1; policy accept;
        ip protocol icmp ct state related counter packets 2 bytes 224 continue
}

```

Figura 6.60 - Aumento del contador related de ICMP

Conclusión del ejemplo:

Este ejemplo, sirve para comprobar, que esto mismo ocurre con conexiones TCP o UDP a cuyo tráfico se le haga *reject*, de forma que el paquete ICMP enviado por PC2, se consideraría RELATED al flujo de comunicación que hayan establecido los protocolos anteriormente, pero la entrada de conntrack nunca llegaría a completarse y permanecería con el flag de UNREPLIED activado.

6.6 Parámetros Conntrack

En este punto se verán las variables que usa conntrack para poder configurar sus distintas opciones.

Estas variables de configuración, se encuentran en la ruta */proc/sys/net/netfilter/* , en la Tabla 16 se nombrará cada una de ellas y se explicará cuál es su uso.

Variable <i>nf_conntrack_</i>	Valores	Descripción
<i>acct</i>	0 - desactivado not 0 - activado (default)	Habilita un contador de paquetes por flow.
<i>buckets</i>	Un número entero	Establece el tamaño de la tabla hash. El tamaño por defecto se calcula dividiendo el tamaño total de memoria entre 16384 para determinar el número de buckets.
<i>checksum</i>	0 - desactivado not 0 - activado (default)	Verifica el checksum de los paquetes y en caso de estar mal, marcará el paquete como inválido.
<i>count</i>	Un número entero	Número de entradas de flujos de comunicación.
<i>events</i>	0 - desactivado not 0 - activado (default)	Habilita el uso de los estados del espacio de usuarios a través de Netlink.
<i>expect_max</i>	Mínimo:1 Máximo:(<i>nf_conntrack_buckets</i> /256)}	Número máximo de buckets de la tabla de expectación.
<i>frag6_high_thresh</i>	262144 (default)	Memoria máxima a usar para reensamblar paquetes Ipv6.
<i>frag6_low_thresh</i>	196608 (default)	Memoria mínima a usar para reensamblar paquetes Ipv6.

<i>frag6_timeout</i>	60 (default - segundos)	Tiempo que se mantiene un fragmento de Ipv6 en memoria.
<i>generic_timeout</i>	600 (default - segundos)	Tiempo de vida de protocolos de capa 4 desconocidos por la aplicación, o bien no son soportados.
<i>helper</i>	0 - desactivado (default) not 0 - activado	Habilita el uso automático de helpers.
<i>icmp_timeout</i>	30 (default - segundos)	Tiempo de vida de una entrada Icmp-timeout.
<i>icmpv6_timeout</i>	30 (default - segundos)	Tiempo de vida de una entrada Icmpv6-timeout.
<i>log_invalid</i>	0 - disable (default) 1 - log ICMP packets 6 - log TCP packets 17 - log UDP packets 33 - log DCCP packets 41 - log ICMPv6 packets 136 - log UDPLITE packets 255 - log packets of any protocol	Realiza el log de los paquetes inválidos según el valor especificado.
<i>max</i>	nf_conntrack_buckets valor * 4	Tamaño máximo de la tabla conntrack.
<i>tcp_be Liberal</i>	0 - desactivado (default) not 0 - activado	Este valor permite ser liberal a la hora de marcar los paquetes como inválido, en caso de que su valor sea distinto de cero, solo marcará como inválido ciertos casos, como puede ser los paquetes con flag RST sin conexión establecida.
<i>tcp_loose</i>	0 - desactivado not 0 - activado (default)	Si su valor es cero, deshabilitamos el tomar entradas preexistentes en conntrack. En todos los casos intenta crear una entrada. Internamente el código permite que esta variable tenga poco uso y solo sea usada en caso de tener firewalls de refuerzo o balanceadores de carga.
<i>tcp_max_retrans</i>	3 (default)	Número máximo de retransmisiones que puede tener un paquete.

<i>tcp_timeout_close</i>	10 (default - segundos)	Tiempo de vida de una entrada con el estado close.
<i>tcp_timeout_close_wait</i>	60 (default - segundos)	Tiempo de vida de una entrada con el estado close_wait.
<i>tcp_timeout_established</i>	432000 (5 días) (default)	Tiempo de vida de una entrada con el estado established.
<i>tcp_timeout_fin_wait</i>	120 (default - segundos)	Tiempo de vida de una entrada con el estado fin_wait.
<i>tcp_timeout_last_ack</i>	30 (default - segundos)	Tiempo de vida de una entrada con el estado last_ack.
<i>tcp_timeout_max_retrans</i>	300 (default - segundos)	Tiempo de vida de una entrada con el estado max_retrans.
<i>tcp_timeout_syn_recv</i>	60 (default - segundos)	Tiempo de vida de una entrada con el estado syn_recv.
<i>tcp_timeout_syn_sent</i>	120 (default - segundos)	Tiempo de vida de una entrada con el estado syn_sent.
<i>tcp_timeout_time_wait</i>	120 (default - segundos)	Tiempo de vida de una entrada con el estado time_wait.
<i>tcp_timeout_unacknowledged</i>	300 (default - segundos)	Tiempo de vida de una entrada con el estado unacknowledged.
<i>timestamp</i>	0 - desactivado (default) not 0 - activado	Habilita el timestamp de las entradas de conntrack.
<i>udp_timeout</i>	30 (default - segundos)	Tiempo de vida de una entrada UDP.
<i>udp_timeout_stream</i>	120 (default - segundos)	Tiempo de vida de una entrada de flujo de intercambio UDP.
<i>gre_timeout</i>	30 (default - segundos)	Tiempo de vida de una entrada GRE.
<i>gre_timeout_stream</i>	180 (default - segundos)	Tiempo de vida de una entrada GRE con flujo de transmisión.

Tabla 16 - Variables configuración conntrack

Si se usa el siguiente comando, se puede ver los valores por defecto de todos los campos explicados anteriormente (ver Figura 6.61).

Equipo: Encaminador	Usuario: root
<pre>\$ sysctl net.netfilter grep conntr</pre>	
<pre>root@localhost:/home/dit# sysctl net.netfilter grep conntr net.netfilter.nf_conntrack_acct = 0 net.netfilter.nf_conntrack_buckets = 16384 net.netfilter.nf_conntrack_checksum = 1 net.netfilter.nf_conntrack_count = 1 net.netfilter.nf_conntrack_dccp_loose = 1 net.netfilter.nf_conntrack_dccp_timeout_closereq = 64 net.netfilter.nf_conntrack_dccp_timeout_closing = 64 net.netfilter.nf_conntrack_dccp_timeout_open = 43200 net.netfilter.nf_conntrack_dccp_timeout_partopen = 480 net.netfilter.nf_conntrack_dccp_timeout_request = 240 net.netfilter.nf_conntrack_dccp_timeout_respond = 480 net.netfilter.nf_conntrack_dccp_timeout_timewait = 240 net.netfilter.nf_conntrack_events = 1 net.netfilter.nf_conntrack_expect_max = 256 net.netfilter.nf_conntrack_generic_timeout = 600 net.netfilter.nf_conntrack_helper = 0 net.netfilter.nf_conntrack_icmp_timeout = 30 net.netfilter.nf_conntrack_log_invalid = 0 net.netfilter.nf_conntrack_max = 65536 net.netfilter.nf_conntrack_sctp_timeout_closed = 10 net.netfilter.nf_conntrack_sctp_timeout_cookie_echoed = 3 net.netfilter.nf_conntrack_sctp_timeout_cookie_wait = 3 net.netfilter.nf_conntrack_sctp_timeout_established = 432000 net.netfilter.nf_conntrack_sctp_timeout_heartbeat_acked = 210 net.netfilter.nf_conntrack_sctp_timeout_heartbeat_sent = 30 net.netfilter.nf_conntrack_sctp_timeout_shutdown_ack_sent = 3 net.netfilter.nf_conntrack_sctp_timeout_shutdown_recd = 0 net.netfilter.nf_conntrack_sctp_timeout_shutdown_sent = 0 net.netfilter.nf_conntrack_tcp_be Liberal = 0 net.netfilter.nf_conntrack_tcp_loose = 1 net.netfilter.nf_conntrack_tcp_max_retrans = 3 net.netfilter.nf_conntrack_tcp_timeout_close = 15 net.netfilter.nf_conntrack_tcp_timeout_close_wait = 60 net.netfilter.nf_conntrack_tcp_timeout_established = 432000 net.netfilter.nf_conntrack_tcp_timeout_fin_wait = 120 net.netfilter.nf_conntrack_tcp_timeout_last_ack = 30 net.netfilter.nf_conntrack_tcp_timeout_max_retrans = 300 net.netfilter.nf_conntrack_tcp_timeout_syn_recv = 60 net.netfilter.nf_conntrack_tcp_timeout_syn_sent = 120 net.netfilter.nf_conntrack_tcp_timeout_time_wait = 120 net.netfilter.nf_conntrack_tcp_timeout_unacknowledged = 300 net.netfilter.nf_conntrack_timestamp = 0 net.netfilter.nf_conntrack_udp_timeout = 30 net.netfilter.nf_conntrack_udp_timeout_stream = 180 root@localhost:/home/dit#</pre>	

Figura 6.61 - Valores parámetros conntrack en Encaminador

6.7 Helpers

Como hemos visto anteriormente, conntrack realiza el seguimiento de flujos de comunicación y para ello, usa ciertos parámetros para poder identificar estos flujos.

Existen protocolos que no solo usan un único canal de comunicación, sino que usan varios, uno de ellos para control y otro para la transferencia de datos (protocolos multi-flow).

Por ello, usamos lo que se denomina “helpers”, que no son más que unos módulos de ayuda que nos permiten asociar el tráfico de dos flujos distintos a una misma comunicación.

Un ejemplo es el protocolo FTP, el cual usa el puerto 21 para conectarse al servidor e intercambiar información de control y el puerto 20 para el intercambio de datos si se usa en modo activo, en modo pasivo, este usa un algoritmo por el cual, calcula el puerto origen y destino siendo estos superiores al puerto 1024.

De esta forma, asocia el flujo de intercambio de datos al de control, que es el primero que sea crea.

Descripción del ejemplo.

En este ejemplo se realizará la prueba de una conexión FTP, en la cual se usará el helper correspondiente para ver el camino de los paquetes y que estados se le asignan a cada uno de estos.

Para ello, se usará el escenario de 2 equipos “Anexo E - Escenario 2 Equipos”.

- PC1 se encargará de realizar la petición al servidor FTP, cuya dirección Ip de la máquina pública es: 35.163.228.146
- Encaminador, será el encargado de realizar el *routing* de los paquetes, así como de monitorizar las reglas y entradas de conntrack.

Análisis del ejemplo.

1. **[Equipo:Encaminador - Usuario:root]** Lo primero que debemos hacer es cargar tanto el helper de FTP como el helper de NAT FTP.
 - El helper de FTP, sirve para poder asociar los dos flujos que se abren (datos y control), a la misma comunicación.
 - El helper de NAT FTP, permite que, aunque se realice NAT de la conexión FTP, podamos ver la conexión en conntrack y no aparezca como *invalid*.

Equipo:Encaminador **Usuario:**root

```
$ modprobe nf_conntrack_ftp  
$ modprobe nf_nat_ftp
```

2. **[Equipo:Encaminador - Usuario:root]** Pasamos a continuación a crear las tablas de nftables que se usarán en este ejemplo.
 - Una tabla nat de tipo ip.
 - Una tabla filter de tipo ip.
 - Una tabla ingress de tipo netdev.

Equipo: Encaminador **Usuario:** root

```
$ nft add table ip nat
$ nft add table ip filter
$ nft add table netdev ingress
```

3. [Equipo:Encaminador - Usuario:root] Se crean los siguientes grupos de cadenas:

- 1) **Tabla filter:** cadenas prerouting, input, forward, postrouting y output.
- 2) **Tabla nat:** cadenas pre y post.
- 3) **Tabla ingress:** cadenas ingr y ens33.

Equipo: Encaminador **Usuario:** root

```
#Cadenas de tabla nat
$ nft add chain 'ip nat pre { type nat hook prerouting priority -10 ; }'
$ nft add chain 'ip nat post { type nat hook postrouting priority 10 ; }'

#Cadenas de tabla filter
$ nft add chain 'ip filter prerouting { type filter hook prerouting priority 0 ; }'
$ nft add chain 'ip filter postrouting { type filter hook postrouting priority 0 ; }'
$ nft add chain 'ip filter input { type filter hook input priority 0 ; }'
$ nft add chain 'ip filter forward { type filter hook forward priority 0 ; }'
$ nft add chain 'ip filter output { type filter hook output priority 0 ; }'

#Cadenas de tabla ingress
$ nft add 'chain netdev ingress ingr { type filter hook ingress device ens38 priority 0 ; }'
$ nft add 'chain netdev ingress ens33 { type filter hook ingress device ens33 priority 0 ; }'
```

4. [Equipo:Encaminador - Usuario:root] Se crean las siguientes reglas:

- 1) En la cadena ingr, se añade una regla que active *nftrace* para todo el tráfico entrante.
- 2) En la cadena ens33, se añade una regla que active *nftrace* para todo el tráfico entrante.
- 3) En la cadena prerouting de filter, se añade una regla que contabilice el tráfico cuyo *state* sea *new*.
- 4) En la cadena prerouting de filter, se añade una regla que contabilice el tráfico cuyo *state* sea *related*.
- 5) En la cadena prerouting de filter, se añade una regla que contabilice el tráfico cuyo *state* sea *established*.
- 6) En la cadena prerouting de filter, se añade una regla que contabilice el tráfico cuyo *state* sea *invalid*.
- 7) En la cadena forward de filter, se añade una regla que contabilice el tráfico cuyo *state* sea *new, established o related*.

- 8) En la cadena postrouting de nat, se añade una regla para contabilizar y realizar el *masquerade* al tráfico cuya Ip origen sea la de PC1.

Equipo:Encaminador **Usuario:**root

```
#Reglas tabla ingress
$ nft add rule netdev ingress ingr meta nftrace set 1 counter
$ nft add rule netdev ingress ens33 meta nftrace set 1 counter

#Reglas tabla filter
$ nft add rule ip filter prerouting ct state new continue
$ nft add rule ip filter prerouting ct state established continue
$ nft add rule ip filter prerouting ct state related continue
$ nft add rule ip filter prerouting ct state invalid continue
$ nft add rule ip 'filter prerouting ct state { new, estbalished,
related } continue'

#Regla tabla nat
$ nft add rule ip nat post ip saddr 10.10.1.20 counter masquerade
```

5. **[Equipo:Encaminador - Usuario:root]** Una vez cargado los módulos y creada la estructura de tablas y cadenas de nftables, pasamos a crear el *Stateful object* de tipo ct helper, que nos ayude a realizar el seguimiento de la conexión FTP.

Equipo:Encaminador **Usuario:**root

```
$ nft add' ct helper ip filter ftp { type "ftp" protocol tcp ; }
```

6. **[Equipo:Encaminador - Usuario:root]** Una vez creado el objeto de tipo helper, pasamos a crear su regla en la cadena forward.

En la cadena forward de filter, se añade una regla que contabilice el tráfico TCP cuyo puerto destino sea el 21 y se le asigna el helper “ftp”, por último, se añade un contador a esta regla.

Equipo:Encaminador **Usuario:**root

```
$ nft add rule ip filter' forward tcp dport ftp ct helper set
"ftp" counter accept '
```

7. **[Equipo:Encaminador - Usuario:root]** Listamos el *ruleset* de nftables.

Podemos ver la salida del comando en las Figura 6.62 y Figura 6.63.

Equipo:Encaminador **Usuario:**root

```
$ nft list ruleset
```

```

root@localhost:/home/dit# nft list ruleset -a
table ip filter {
    ct helper ftp {
        type "ftp" protocol tcp
        l3proto ip
    }

    chain prerouting {
        type filter hook prerouting priority 0; policy accept;
        ct state new counter packets 50 bytes 5986 continue # handle 7
        ct state related counter packets 0 bytes 0 continue # handle 8
        ct state established counter packets 1 bytes 328 continue # handle 9
        ct state invalid counter packets 0 bytes 0 continue # handle 10
    }

    chain postrouting {
        type filter hook postrouting priority 0; policy accept;
    }

    chain input {
        type filter hook input priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
        ct state { established, related, new } counter packets 0 bytes 0 continue # handle 14
        tcp dport ftp ct helper set "ftp" counter packets 0 bytes 0 accept # handle 15
    }
}

```

Figura 6.62 - Reglas para ct ftp-1

```

table ip nat {
    chain pre {
        type nat hook prerouting priority -10; policy accept;
    }

    chain post {
        type nat hook postrouting priority 10; policy accept;
        ip saddr 10.10.1.20 counter packets 0 bytes 0 masquerade # handle 4
    }
}

table netdev ingress {
    chain ingr {
        type filter hook ingress device ens38 priority 0; policy accept;
        nftrace set 1 counter packets 33 bytes 3783 # handle 5
    }

    chain ens33 {
        type filter hook ingress device ens33 priority 0; policy accept;
        nftrace set 1 counter packets 62 bytes 6100 # handle 6
    }
}

```

Figura 6.63 - Reglas para ct ftp-2

8. [Equipo:Encaminador - Usuario:root] Se abren dos terminales en el equipo Encaminador:

- Terminal 1: Se abre conntrack en modo monitor de eventos, usando la opción de mostrar todos los eventos.
- Terminal 2: Se abre un monitor de reglas nftables.
- Terminal 3: Se abre un monitor de conntrack para visualizar la tabla expect, en la cual, se añadirá la entrada de expectación de la comunicación con FTP.

Equipo:Encaminador Usuario:root

```

#Terminal 1
$ conntrack -E --event-mask ALL

#Terminal 2
$ nft monitor trace

#Terminal 3
$ nft conntrack -E expect

```

9. [Equipo:PC1 - Usuario:root] Pasamos a realizar la conexión con el servidor FTP desde PC1, para ello usaremos el siguiente comando.

Se indican además las credenciales para conectar con el servidor FTP.

Una vez conectado al servidor, se procederá a realizar la subida de un fichero, en nuestro caso, usamos un index.html de prueba.

```
Equipo:PC1 Usuario:root
```

```
$ ftp ftp.dlpitest.com
# Usuario: dlpuser
# Password: rNrKYTX9g7z3RgJRxWuGHbeu

#Ya nos encontramos conectados al servidor, ahora subimos el
archivo

$put index.html
```

Podemos observar en la Figura 6.64, la salida de los comandos anteriormente mencionados, así como de la conexión en modo pasivo que se realiza para subir el archivo index.html.

```
[root@localhost dit]# ftp ftp.dlpitest.com
Connected to ftp.dlpitest.com (35.163.228.146).
220 Help support DLP Test https://commerce.coinbase.com/checkout/0799aa1f-1e3f-4
0c7-a19b-e18a1e1913e7
Name (ftp.dlpitest.com:root): dlpuser
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put index.html
local: index.html remote: index.html
227 Entering Passive Mode (35,163,228,146,4,24).          ↑↑
150 Ok to send data.
226 Transfer complete.
13796 bytes sent in 0.0025 secs (5520.61 Kbytes/sec)
ftp> exit
221 Goodbye.
[root@localhost dit]#
```

Figura 6.64 - Conexión FTP desde PC1

Como se puede ver en la Figura 6.64, se han marcado dos números de la conexión en modo pasivo.

Estos números, son los que indican el puerto que abre el servidor FTP. Para obtener el valor del puerto se hace la siguiente operación ($4 * 256 + 24 = 1048$), por lo que ya sabremos que la conexión con el servidor, se realizará en el puerto 1048.

10. [Equipo:Encaminador - Usuario:root] En Encaminador, observamos la salida del monitor de *nftrace* para ver como se establece la conexión del flujo de control.

A continuación, veremos los tres paquetes TCP que utiliza PC1 para establecer la conexión de control con el servidor FTP.

```

trace id 1ba7c53a netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether da
ddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 35.163.228.146 ip dscp cs0 ip ecn not-ect ip
ttl 64 ip id 18737 i
p length 60 tcp sport 34598 tcp dport ftp tcp flags == syn tcp window 29200
trace id 1ba7c53a netdev ingress ingr rule nftrace set 1 counter packets 1515 bytes 100102 (verd
ict continue)
trace id 1ba7c53a netdev ingress ingr verdict continue
trace id 1ba7c53a netdev ingress ingr
trace id 1ba7c53a ip nat pre verdict continue
trace id 1ba7c53a ip nat pre
trace id 1ba7c53a ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether d
addr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 35.163.228.146 ip dscp cs0 ip ecn not-ect ip
ttl 64 ip id 18737
ip length 60 tcp sport 34598 tcp dport ftp tcp flags == syn tcp window 29200
trace id 1ba7c53a ip filter prerouting rule ct state new counter packets 201 bytes 16866 continu
e (verdict continue)
trace id 1ba7c53a ip filter prerouting verdict continue
trace id 1ba7c53a ip filter prerouting
trace id 1ba7c53a ip filter forward packet: iif "ens38" oif "ens33" ether saddr 00:0c:29:62:ed:a
d ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 35.163.228.146 ip dscp cs0 ip ecn n
ot-ect ip ttl 63 ip
id 18737 ip length 60 tcp sport 34598 tcp dport ftp tcp flags == syn tcp window 29200
trace id 1ba7c53a ip filter forward rule ct state { } counter packets 2917 bytes 19186132 continu
e (verdict continue)
trace id 1ba7c53a ip filter forward rule tcp dport ftp ct helper set "ftp" counter packets 18 by
tes 829 accept (verdict accept)
trace id 1ba7c53a ip filter postrouting verdict continue
trace id 1ba7c53a ip filter postrouting

```

Figura 6.65 - Paquete TCP SYN flujo control FTP

En la Figura 6.65, se puede ver el camino del paquete TCP SYN enviado desde PC1 al servidor FTP:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena *ingr*, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en *ingr* es *continue*, por lo que el paquete continúa su camino hacia la cadena *prerouting* de *filter*.
- 4) En *prerouting* se procesa y en una fase temprana se clasifica el paquete con *state NEW*, su veredicto es *continue*, por lo que se continúa con su procesado.
- 5) Se consulta el encaminamiento del paquete, determinando que este ha de ser encaminado hacia el servidor FTP.
- 6) Al tener que ser encaminado, el paquete continúa su camino por el *hook forward*, donde hace *match* con la regla que inspecciona si el paquete tiene el *state NEW*, ESTABLISHED o RELATED. Además, hace *match* con la segunda regla, en la cual se activa el uso del helper de FTP para este flujo de comunicación y siendo el veredicto de esta cadena *continue*.
- 7) El paquete llega al *hook postrouting*, donde, antes de abandonar el equipo se le aplica la regla de NAT y se le realiza *masquerade*, cambiando su Ip origen por la de la interfaz “ens33” del equipo Encaminador.

En este primer paquete, se puede ver que el puerto usado por PC1 es el 34598, mientras que el del servidor es el 21.

```

trace id e82b55a1 netdev ingress ens33 packet: iif "ens33" ether saddr 00:50:56:f8:b6:0f ether d
addr 00:0c:29:f0:01:14 ip saddr 35.163.228.146 ip daddr 192.168.106.137 ip dscp cs0 ip ecn not-e
ct ip ttl 128 ip id
65456 ip length 44 [tcp sport ftp tcp dport 34598] [tcp flags == 0x12] [tcp window 64240]
trace id e82b55a1 netdev ingress ens33 rule ntrace set 1 counter packets 1606 bytes 19112238 (v
erdict continue)
trace id e82b55a1 netdev ingress ens33 verdict continue
trace id e82b55a1 netdev ingress ens33
trace id e82b55a1 ip filter prerouting packet: iif "ens33" ether saddr 00:50:56:f8:b6:0f ether d
addr 00:0c:29:f0:01:14 ip saddr 35.163.228.146 ip daddr 10.10.1.20 ip dscp cs0 ip ecn not-ect ip
ttl 128 ip id 65456
    ip length 44 tcp sport ftp tcp dport 34598 [tcp flags == 0x12] [tcp window 64240]
trace id e82b55a1 ip filter prerouting rule ct state established [counter packets 2787 bytes 1917
8589 continue (verdict continue)]
trace id e82b55a1 ip filter prerouting verdict continue
trace id e82b55a1 ip filter prerouting
trace id e82b55a1 ip filter forward packet: iif "ens33" oif "ens38" ether saddr 00:50:56:f8:b6:0
f ether daddr 00:0c:29:f0:01:14 ip saddr 35.163.228.146 ip daddr 10.10.1.20 ip dscp cs0 ip ecn n
ot-ect ip ttl 127 ip
    id 65456 ip length 44 tcp sport ftp tcp dport 34598 tcp flags == 0x12 tcp window 64240
trace id e82b55a1 ip filter forward rule ct state { } counter packets 2917 bytes 19186132 conti
nue (verdict continue)
trace id e82b55a1 ip filter forward verdict continue
trace id e82b55a1 ip filter forward
trace id e82b55a1 ip filter postrouting verdict continue
trace id e82b55a1 ip filter postrouting

```

Figura 6.66 - Paquete TCP SYN-ACK flujo control FTP

En la Figura 6.66, se puede ver el camino del paquete TCP SYN-ACK enviado desde el servidor FTP a PC1:

- 1) El paquete sale de la máquina pública(servidor FTP) y llega a Encaminador por la interfaz “ens33”
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena ens33, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en ens33 es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 4) En prerouting se procesa y en una fase temprana se clasifica el paquete con *state* ESTABLISHED, su veredicto es *continue*, por lo que se continúa con su procesado.
- 5) Se consulta el encaminamiento del paquete, determinando que este ha de ser encaminado hacia PC1.
- 6) Al tener que ser encaminado, el paquete continúa su camino por el *hook forward*, donde hace *match* con la regla que inspecciona si el paquete tiene el *state* NEW, ESTABLISHED o RELATED.

Además, hace *match* con la segunda regla, en la cual se activa el uso del helper de FTP para este flujo de comunicación y siendo el veredicto de esta cadena *continue*.

- 7) El paquete llega al *hook postrouting* y sale de Encaminador hacia PC1 por la interfaz “ens38”.

```

trace id 17c42f5b netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 35.163.228.146 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 18738
p length 40 tcp sport 34598 tcp dport ftp [tcp flags == ack] [tcp window 29200]
trace id 17c42f5b netdev ingress ingr rule nftrace set 1 counter packets 1515 bytes 100102 (verdict continue)
trace id 17c42f5b netdev ingress ingr verdict continue
trace id 17c42f5b netdev ingress ingr
trace id 17c42f5b ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 35.163.228.146 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 18738
ip length 40 [tcp sport 34598] [tcp dport ftp] [tcp flags == ack] [tcp window 29200]
trace id 17c42f5b ip filter prerouting rule [ct state established] counter packets 2787 bytes 19178
589 continue (verdict continue)
trace id 17c42f5b ip filter prerouting verdict continue
trace id 17c42f5b ip filter prerouting
trace id 17c42f5b ip filter forward packet: iif "ens38" oif "ens33" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 35.163.228.146 ip dscp cs0 ip ecn not-ect ip ttl 63 ip id 18738 ip length 40 tcp sport 34598 tcp dport ftp [tcp flags == ack] [tcp window 29200]
trace id 17c42f5b ip filter forward rule ct state { } counter packets 2917 bytes 19186132 continue (verdict continue)
trace id 17c42f5b ip filter forward rule [tcp dport ftp] [ct helper set "ftp"] counter packets 18 bytes 829 accept (verdict accept)
trace id 17c42f5b ip filter postrouting verdict continue
trace id 17c42f5b ip filter postrouting

```

Figura 6.67 - Paquete TCP ACK flujo control FTP

En la Figura 6.67, se puede ver el camino del paquete TCP ACK enviado desde PC1 al servidor FTP, y con el cual, queda establecida la conexión de control entre ambos equipos:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena *ingr*, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en *ingr* es *continue*, por lo que el paquete continúa su camino hacia la cadena *prerouting* de *filter*.
- 4) En *prerouting* se procesa y en una fase temprana se clasifica el paquete con *state ESTABLISHED*, su veredicto es *continue*, por lo que se continúa con su procesado.
- 5) Se consulta el encaminamiento del paquete, determinando que este ha de ser encaminado hacia el servidor FTP.
- 6) Al tener que ser encaminado, el paquete continua su camino por el *hook forward*, donde hace *match* con la regla que inspecciona si el paquete tiene el *state NEW*, *ESTABLISHED* o *RELATED*.

Además, hace *match* con la segunda regla, en la cual se activa el uso del helper de FTP para este flujo de comunicación y siendo el veredicto de esta cadena *continue*.

- 8) El paquete llega al *hook postrouting* y sale de Encaminador hacia el servidor FTP por la interfaz “ens33”.

11. [Equipo:Encaminador - Usuario:root] En Encaminador, observamos la salida del monitor de conntrack para ver como se han creado las entradas del flujo de control.

```

[NEW] tcp      6 120 SYN SENT src=10.10.1.20 dst=35.163.228.146 sport=34598 dport=21 [UNREPLIED] src=35.163.228.146 dst=192.168.106.137 sport=21 dport=34598 helper=ftp
[UPDATE] tcp      6 60 SYN RECV src=10.10.1.20 dst=35.163.228.146 sport=34598 dport=21 src=35.163.228.146 dst=192.168.106.137 sport=21 dport=34598 helper=ftp
[UPDATE] tcp      6 432000 ESTABLISHED src=10.10.1.20 dst=35.163.228.146 sport=34598 dport=21 src=35.163.228.146 dst=192.168.106.137 sport=21 dport=34598 [ASSURED] helper=ftp

```

Figura 6.68 - Entradas conntrack conexión flujo de control FTP.

En la Figura 6.68, vemos como se han creado las entradas del flujo de control de FTP.

Estas entradas tienen en común, que usan el helper ftp.

- 1) La primera entrada con el estado TCP SYN_SENT, se crea cuando el paquete TCP SYN enviado por PC1 hacia el servidor FTP, cruza el *hook postrouting* para salir del equipo Encaminador. Esta entrada tiene activado el flag UNREPLIED.
- 2) La entrada se actualiza con el estado TCP SYN_RECV, cuando el paquete TCP SYN-ACK enviado por el servidor FTP hacia PC1, cruza el *hook postrouting* para salir del equipo Encaminador. Se elimina también el flag UNREPLIED de la entrada.
- 3) Por último, la entrada se vuelve a actualizar con el estado TCP ESTABLISHED, cuando el paquete TCP ACK enviado por PC1 hacia el servidor FTP cruza el *hook postrouting*, para salir del equipo Encaminador. En esta entrada se marca la conexión con el flag ASSURED.

Antes de pasar a ver el siguiente punto, se debe explicar que es una “expectación”[55] y como funciona.

En este ejemplo, como se ha visto en la Figura 6.64, se comunica que puerto se va a utilizar para el flujo de datos de la comunicación FTP.

Nosotros hemos añadido el helper a las reglas que tengan como puerto TCP destino el 21.

Al comunicar el puerto usado para el flujo de datos, conntrack crea en su tabla expect, una entrada de expectación.

Este tipo de entradas, es usada para las conexiones que usan helper, es decir, aquellas que son multi-flow.

En esta expectación, indicamos que se espera la creación de una entrada, que guardará relación con una entrada ya existente en conntrack (esta entrada es conocida como entrada maestra) y de la que solo conocemos algunos datos.

Una vez se ve el primer paquete que coincide con los campos de la entrada de expectación, esta es borrada y se crea una nueva entrada en conntrack, que estará relacionada, con la entrada denominada maestra.

A continuación, se verán las partes de una entrada *expectation*.

```
root@localhost:/home/dit# conntrack -E expect
[NEW] 300 proto=6 src=10.10.1.20 dst=35.163.228.146 sport=0 dport=1035 mask-src=255.255.255.255 mask-dst=255.255.255.255
      sport=0 dport=65535 master-src=10.10.1.20 master-dst=35.163.228.146 sport=43452 dport=21 class=0 helper=ftp
```

Figura 6.69 - Campos de una entrada expectation conntrack

Analizamos la información mostrada en la Figura 6.69 (Entrada conntrack expectation):

- 1) Para comenzar, se tiene el estado de la entrada conntrack, en este caso NEW.
- 2) Valor del *timeout* de ese tipo de expectación, el valor por defecto para una expectación FTP es de 300 segundos.
- 3) Información decimal del protocolo usado, como se puede ver, el valor 6 hace referencia al protocolo TCP.
- 4) Dirección origen de la expectación, en esta figura vemos que se espera que el origen sea la Ip de PC1.
- 5) Dirección destino de la expectación, en esta figura vemos que se espera que el destino sea la Ip del servidor FTP.
- 6) Puerto origen de la expectación, el 0 que se puede ver, significa que puede usarse cualquier puerto origen.

- 7) Puerto destino de la expectación, en este ejemplo su valor es 1035.
 - 8) Máscara de la dirección origen de la expectación, la cual nos indica el valor de coincidencia que ha de tener con la dirección origen mostrada, en este caso, al ser una máscara 255.255.255.255, nos indica que la dirección origen del paquete que cumpla la expectación, debe ser la dada en el campo dirección origen.
 - 9) Máscara de la dirección destino de la expectación, la cual nos indica el valor de coincidencia que ha de tener con la dirección destino mostrada, en este caso, al ser una máscara 255.255.255.255, nos indica que la dirección destino del paquete que cumpla la expectación, debe ser la dada en el campo dirección destino.
 - 10) Máscara del puerto origen, la cual nos indica la coincidencia que se desea con el puerto origen del paquete que cumpla con la expectación, el valor 0 hace referencia a que cualquier puerto es válido.
 - 11) Máscara del puerto destino, la cual nos indica la coincidencia que se desea con el puerto destino del paquete que cumpla con la expectación, el valor 65535 hace referencia a que la coincidencia debe ser exacta con el valor del campo puerto destino.
 - 12) Dirección origen de la entrada maestra a la que se relaciona la expectación.
 - 13) Dirección destino de la entrada maestra a la que se relaciona la expectación.
 - 14) Puerto origen de la entrada maestra a la que se relaciona la expectación.
 - 15) Puerto destino de la entrada maestra a la que se relaciona la expectación.
 - 16) La clase, es un valor que sirve para identificar los diferentes tipos de *helpers* usados.
- Este valor solo se usa para depuración y no tiene valor desde el punto de vista de un administrador de redes.
- 17) Helper usado, en este caso, indica que se ha usado un helper FTP.

Tabla de parámetros de una entrada conntrack		
1º	<i>Estado de la entrada conntrack</i>	NEW
2º	<i>Timeout de la entrada conntrack</i>	300 segundos
3º	<i>Valor decimal del protocolo de capa 4</i>	6
4º	<i>IP origen (expectation)</i>	10.10.1.20
5º	<i>IP destino (expectation)</i>	35.163.228.146
6º	<i>Puerto origen (expectation)</i>	0
7º	<i>Puerto destino (expectation)</i>	1034
8º	<i>Máscara IP origen (expectation)</i>	255.255.255.255
9º	<i>Máscara IP destino (expectation)</i>	255.255.255.255
10º	<i>Máscara puerto origen</i>	0

	<i>(expectation)</i>	
11º	Máscara puerto destino <i>(expectation)</i>	65535
12º	IP origen (entrada maestra)	10.10.1.20
13º	IP destino (entrada maestra)	35.163.228.146
14º	Puerto origen (entrada maestra)	43452
15º	Puerto destino (entrada maestra)	21
16º	Class del protocolo usado	0
17º	Helper usado	ftp

Tabla 17 - Campos de la entrada conntrack expectation

La Tabla 17, recoge los distintos campos con la información mostrada en la Figura 6.69.

12. [Equipo:Encaminador - Usuario:root] En el equipo Encaminador, observamos la salida del monitor de *nftrace*, en el cual se puede observar cómo se establece la conexión del flujo de datos.

```
trace id abadc8a2 netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether dad dr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 35.163.228.146 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 6332 ip length 60 tcp sport 52733 tcp dport 1048 tcp flags == syn tcp window 29200
trace id abadc8a2 netdev ingress ingr rule ntrace set 1 counter packets 1515 bytes 100102 (verdict continue)
trace id abadc8a2 netdev ingress ingr verdict continue
trace id abadc8a2 netdev ingress ingr
trace id abadc8a2 ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether dad dr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 35.163.228.146 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 6332 ip length 60 tcp sport 52733 tcp dport 1048 tcp flags == syn tcp window 29200
trace id abadc8a2 ip filter prerouting rule ct state related counter packets 1 bytes 60 continue (verdict continue)
trace id abadc8a2 ip filter prerouting verdict continue
trace id abadc8a2 ip filter prerouting
trace id abadc8a2 ip filter forward packet: iif "ens38" oif "ens33" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 35.163.228.146 ip dscp cs0 ip ecn not-ect ip ttl 63 ip id 6332 ip length 60 tcp sport 52733 tcp dport 1048 tcp flags == syn tcp window 29200
trace id abadc8a2 ip filter forward rule ct state { } counter packets 2917 bytes 19186132 continue (verdict continue)
trace id abadc8a2 ip filter forward verdict continue
trace id abadc8a2 ip filter forward
trace id abadc8a2 ip filter postrouting verdict continue
trace id abadc8a2 ip filter postrouting
```

Figura 6.70 - Paquete TCP SYN flujo datos FTP

En la Figura 6.70, se puede ver el camino del paquete TCP SYN enviado desde PC1 al servidor FTP, para establecer la conexión del flujo de datos:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena *ingr*, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en la cadena *ingr* es *continue*, por lo que el paquete continúa su camino hacia la

cadena prerouting de filter.

- 4) En prerouting, se procesa y en una fase temprana se clasifica el paquete con *state RELATED*, ya que como se verá más adelante, este paquete cumple con la entrada *expectation* de conntrack. Así, su estado es *related* al flujo de comunicación que ya existía en conntrack. Su veredicto es *continue*, por lo que se continúa con su procesado.
- 5) Se consulta el encaminamiento del paquete, determinando que este ha de ser encaminado hacia el servidor FTP.
- 6) Al tener que ser encaminado, el paquete continúa su camino por el *hook forward*, donde hace *match* con la regla que inspecciona si el paquete tiene el *state NEW, ESTABLISHED o RELATED*.
- 7) El paquete llega al *hook postrouting* y sale de Encaminador hacia el servidor FTP por la interfaz “ens33”.

```
trace id 4c6483f6 netdev ingress ens33 rule nftrace set 1 counter packets 1606 bytes 19112238 (verdict continue)
trace id 4c6483f6 netdev ingress ens33 verdict continue
trace id 4c6483f6 netdev ingress ens33
trace id 4c6483f6 ip filter prerouting packet: iif "ens33" ether saddr 00:50:56:f8:b6:0f ether daddr 00:0c:29:f0:01:14 ip saddr 35.163.228.146 ip daddr 10.10.1.20 ip dscp cs0 ip ecn not-ect ip ttl 128 ip id 65468 ip length 44 [tcp sport 1048 tcp dport 52733] tcp flags == 0x12 tcp window 64240
trace id 4c6483f6 ip filter prerouting rule [ct state established] counter packets 2787 bytes 19178589 continue (verdict continue)
trace id 4c6483f6 ip filter prerouting verdict continue
trace id 4c6483f6 ip filter prerouting
trace id 4c6483f6 ip filter forward packet: iif "ens33" oif "ens38" ether saddr 00:50:56:f8:b6:0f ether daddr 00:0c:29:f0:01:14 [ip saddr 35.163.228.146 ip daddr 10.10.1.20] ip dscp cs0 ip ecn not-ect ip ttl 127 ip id 65468 ip length 44 tcp sport 1048 tcp dport 52733 tcp flags == 0x12 tcp window 64240
trace id 4c6483f6 ip filter forward rule ct state { } counter packets 2917 bytes 19186132 continue (verdict continue)
trace id 4c6483f6 ip filter forward verdict continue
trace id 4c6483f6 ip filter forward
trace id 4c6483f6 ip filter postrouting verdict continue
trace id 4c6483f6 ip filter postrouting
```

Figura 6.71 - Paquete TCP SYN-ACK flujo datos FTP

En la Figura 6.71, se puede ver el camino del paquete TCP SYN-ACK enviado desde el servidor FTP a PC1 (flujo de datos):

- 1) El paquete sale de la máquina pública (servidor FTP) y llega a Encaminador por la interfaz “ens33”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena ens33, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en la cadena ens33 es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 4) En prerouting, se procesa y en una fase temprana se clasifica el paquete con *state ESTABLISHED*, su veredicto es *continue*, por lo que se continúa con su procesado.
- 5) Se consulta el encaminamiento del paquete, determinando que este ha de ser encaminado hacia PC1.
- 6) Al tener que ser encaminado, el paquete continúa su camino por el *hook forward*, donde hace *match* con la regla que inspecciona si el paquete tiene el *state NEW, ESTABLISHED o RELATED*.
- 7) El paquete llega al *hook postrouting* y sale de Encaminador hacia PC1 por la interfaz “ens38”.

```

trace id f58dc31b netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 35.163.228.146 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 6333 ip length 40 tcp sport 52733 tcp dport 1048 tcp flags == ack tcp window 29200
trace id f58dc31b netdev ingress ingr rule nftrace set 1 counter packets 1515 bytes 100102 (verdict continue)
trace id f58dc31b netdev ingress ingr verdict continue
trace id f58dc31b ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 35.163.228.146 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 6333 ip length 40 tcp sport 52733 tcp dport 1048 tcp flags == ack tcp window 29200
trace id f58dc31b ip filter prerouting rule ct state established counter packets 2787 bytes 19178589 continue (verdict continue)
trace id f58dc31b ip filter prerouting verdict continue
trace id f58dc31b ip filter prerouting
trace id f58dc31b ip filter forward packet: iif "ens38" oif "ens33" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 35.163.228.146 ip dscp cs0 ip ecn not-ect ip ttl 63 ip id 6333 ip length 40 tcp sport 52733 tcp dport 1048 tcp flags == ack tcp window 29200
trace id f58dc31b ip filter forward rule ct state { } counter packets 2917 bytes 19186132 continue (verdict continue)
trace id f58dc31b ip filter forward verdict continue
trace id f58dc31b ip filter forward
trace id f58dc31b ip filter postrouting verdict continue
trace id f58dc31b ip filter postrouting

```

Figura 6.72 - Paquete TCP ACK flujo datos FTP

En la Figura 6.72, se puede ver el camino del paquete TCP ACK enviado desde PC1 al servidor FTP, para terminar con el establecimiento de la conexión del flujo de datos:

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena *ingr*, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en *ingr* es *continue*, por lo que el paquete continúa su camino hacia la cadena *prerouting* de *filter*.
- 4) En *prerouting*, se procesa y en una fase temprana se clasifica el paquete con *state ESTABLISHED*, su veredicto es *continue*, por lo que se continúa con su procesado.
- 5) Se consulta el encaminamiento del paquete, determinando que este ha de ser encaminado hacia el servidor FTP.
- 6) Al tener que ser encaminado, el paquete continúa su camino por el *hook forward*, donde hace *match* con la regla que inspecciona si el paquete tiene el *state NEW*, *ESTABLISHED* o *RELATED*.
- 7) El paquete llega al *hook postrouting* y sale de Encaminador hacia el servidor FTP por la interfaz “ens33”.

13. [Equipo:Encaminador - Usuario:root] En Encaminador, observamos la salida de *conntrack* para ver las dos entradas creadas, una perteneciente al flujo de control y otra al flujo de datos.

```

tcp      6 108 TIME WAIT src=10.10.1.20 dst=35.163.228.146 sport=34598 dport=21 src=35.163.228.146 dst=192.168.106.137 sport=21 dport=34598 [ASSURED] mark=0 helper=ftp use=2
tcp      6 106 TIME WAIT src=10.10.1.20 dst=35.163.228.146 sport=52733 dport=1048 src=35.163.228.146 dst=192.168.106.137 sport=1048 dport=52733 [ASSURED] mark=0 use=1
conntrack v1.4.6 (conntrack-tools): 3 flow entries have been shown.

```

Figura 6.73 - Entradas conntrack ejemplo FTP

En la Figura 6.73, se puede observar como *conntrack* tiene registrada dos entradas:

- En la primera de ella, podemos ver los datos de la comunicación de control de FTP, en la cual el puerto destino es el 21, se está usando el helper *ftp* y en el campo *use*, vemos que aparece un 2.

Este valor, nos indica que esta entrada está siendo usada por otra entrada que depende de ella (recordamos que el valor por defecto del campo *use* de una entrada en conntrack es 1).

- En la segunda entrada, podemos ver la entrada del flujo de datos FTP, en la cual el puerto destino es el que se acordó en el modo pasivo (puerto 1048)

Debemos ver también la salida del monitor de conntrack de la tabla expect.

```
root@localhost:/home/dit# conntrack -E expect
[NEW] 300 proto=6 src=10.10.1.20 dst=35.163.228.146 sport=0 dport=1048 mask-src=255.255.255.255
mask-dst=255.255.255.255 sport=0 dport=65535 master-src=10.10.1.20 master-dst=35.163.228.146 sport=3
4598 dport=21 class=0 helper=ftp
[DESTROY] 300 proto=6 src=10.10.1.20 dst=35.163.228.146 sport=0 dport=1048 mask-src=255.255.255.255
mask-dst=255.255.255.255 sport=0 dport=65535 master-src=10.10.1.20 master-dst=35.163.228.146 sport=3
4598 dport=21 class=0 helper=ftp
^Conntrack v1.4.6 (conntrack-tools): 2 expectation events have been shown.
root@localhost:/home/dit#
```

Figura 6.74 - Entrada conntrack en la tabla expect, ejemplo FTP

En la Figura 6.74, podemos ver dos entradas en el monitor de conntrack de la tabla expect.

- La primera entrada, representa la entrada de expectación creada al usar el modo pasivo. En ella, se puede ver que la Ip origen corresponde con PC1, la Ip destino corresponde con el servidor FTP y que el puerto destino es el 1048, puerto que ha sido calculado en la Figura 6.64.

Podemos ver también como la conexión a la que hace referencia (conexión maestra), es la que registró conntrack al finalizar el proceso de establecimiento de conexión del flujo de control.

- La segunda entrada que se puede ver, nos indica la destrucción de la entrada de expectación. Esto se produce porque, al ver el primer paquete que coincide con los datos de la entrada de expectación (en este ejemplo es el paquete TCP SYN del flujo de datos), conntrack destruye esta entrada y crea una nueva entrada en la tabla conntrack.

Conclusión del ejemplo:

Los principales aportes de este ejemplo para un administrador de red son los siguientes:

- La tabla resumen de los campos de una entrada en la tabla *expect* de conntrack, ya que no existe información en la red acerca de los campos de la misma, ni de los valores que estos pueden tomar.
- La comprensión del *state related*, existen muchas inexactitudes a la hora de tratar este estado y en este ejemplo se ha buscado aclarar el funcionamiento del mismo, mostrando que no es más que una especie de *state new*, pero haciendo referencia a entradas de flujos de comunicación que dependen de un flujo de comunicación previo.

Usando también el ejemplo para comprobar, como se realiza el establecimiento de conexión y comunicación de un protocolo ampliamente usado, FTP.

- Este no es el único protocolo que utiliza dos puertos para comunicarse (uno de datos y otro de control), nftables permite usar helpers para los protocolos mostrados en la Tabla 18:

Protocolos con helper en conntrack		
FTP	SIP	SANE
TFTP	H.323	Amanda
NetBios	SNMP	
IRC	PPTP	

Tabla 18 - Protocolos con helper en conntrack

6.8 Pruebas escenario con módulo ct (Connection Tracking)

En este apartado, se verán dos de las opciones más interesantes y usadas del módulo conntrack. La primera de ellas es la marca de conexión, mientras que la segunda ya ha sido usada a lo largo de este capítulo, siendo esta el estado del paquete.

La opción *mark*, nos permite marcar flujos de conexión, pudiendo identificar aquellos que resulten más interesantes para su estudio.

Mientras que la opción de *state*, nos permite saber el estado en el que se categoriza un paquete con respecto a un flujo de conexión.

6.8.1 Ejemplo mark

Descripción del ejemplo:

Para realizar esta prueba, se usará el escenario de 2 equipos “Anexo E - Escenario 2 Equipos”.

El equipo Encaminador, será el que aplique las reglas de nftables, así como observar los flujos de connection tracking, mientras que el PC1 será el encargado de generar tráfico para poder trabajar con él.

En este primer ejemplo, vamos a ver el uso de la marca del módulo conntrack.

Esta marca nos permite asignar una especie de etiqueta al tráfico elegido, para en *hooks* posteriores, aceptarlo, denegarlo o darle un tratamiento distinto.

Vamos a marcar el flujo de conexión que tenga como destino nuestro equipo, con una marca de 0x2 y vamos a marcar el tráfico al que Encaminador tenga que hacerle *forwarding*, con la marca de 0x3.

En postrouting, permitiremos salir al tráfico marcado con 0x2, pero no así al que tenga la marca 0x3.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Pasamos a continuación a crear las tablas de nftables que se usarán en este ejemplo.
 - 1) Una tabla nat de tipo ip.
 - 2) Una tabla filter de tipo ip.

Equipo: Encaminador **Usuario:** root

```
$ nft add table ip nat
$ nft add table ip filter
$ nft add table netdev filteringress
```

2. [Equipo:Encaminador - Usuario:root] Se crean los siguientes grupos de cadenas:

- 1) **Tabla filter:** cadenas prerouting, input, forward, postrouting y output.
- 2) **Tabla nat:** cadenas prerouting y postrouting.

Equipo: Encaminador **Usuario:** root

```
#Cadenas de tabla nat
$ nft add chain 'ip nat prerouting { type nat hook prerouting priority -10 ; }'
$ nft add chain 'ip nat postrouting { type nat hook postrouting priority 10 ; }'

#Cadenas de tabla filter
$ nft add chain 'ip filter prerouting { type filter hook prerouting priority 0 ; }'
$ nft add chain 'ip filter postrouting { type filter hook postrouting priority 0 ; }'
$ nft add chain 'ip filter input { type filter hook input priority 0 ; }'
$ nft add chain 'ip filter forward { type filter hook forward priority 0 ; }'
$ nft add chain 'ip filter output { type filter hook output priority 0 ; }'
```

3. [Equipo:Encaminador - Usuario:root] Se crean las siguientes reglas en nftables:

- 1) En la cadena postrouting de filter, se añade una regla que realice el *drop* del tráfico con el valor de marca 0x3.
- 2) En la cadena postrouting de filter, se añade una regla que realice el *accept* del tráfico con el valor de marca 0x2.
- 3) En la cadena input de filter, se añade una regla que marque el tráfico con el valor 0x2.
- 4) En la cadena forward de filter, se añade una regla que marque el tráfico con el valor 0x3.
- 5) En la cadena postrouting de nat, se añade una regla para contabilizar y realizar el *masquerade* al tráfico cuya Ip origen sea la de PC1.

Equipo: Encaminador **Usuario:** root

```
#Reglas tabla filter
$ nft add rule ip filter postrouting ct mark 0x3 drop
$ nft add rule ip filter postrouting ct mark 0x2 accept
$ nft add rule ip filter input ip protocol icmp ct mark set 0x2
$ nft add rule ip filter forward ip protocol icmp ct mark set 0x3

#Regla tabla nat
```

```
$ nft add rule ip nat post ip saddr 10.10.1.20 counter
masquerade
```

4. [Equipo:Encaminador - Usuario:root] Se muestra el *ruleset* de nftables.

Podemos comprobar la salida del siguiente comando en la Figura 6.75

Equipo:Encaminador Usuario:root

```
$ nft list ruleset -a
```

```
root@localhost:/home/dit# nft list ruleset -a
table ip nat {
    chain postrouting {
        type nat hook postrouting priority 100; policy accept;
        ip saddr 10.10.1.0/24 counter packets 1531 bytes 99164 masquerade # handle 4
    }

    chain prerouting {
        type nat hook prerouting priority 0; policy accept;
        ip daddr 192.168.106.128 dnat to 10.10.1.20 # handle 7
    }
}
table ip filter {
    chain prerouting {
        type filter hook prerouting priority 10; policy accept;
    }

    chain postrouting {
        type filter hook postrouting priority 110; policy accept;
        ct mark 0x00000003 counter packets 0 bytes 0 drop # handle 11
        ct mark 0x00000002 counter packets 0 bytes 0 accept # handle 12
    }

    chain output {
        type filter hook output priority 0; policy accept;
    }

    chain input {
        type filter hook input priority 0; policy accept;
        ip protocol icmp ct mark set 0x00000002 counter packets 0 bytes 0 # handle 9
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
        ip protocol icmp ct mark set 0x00000003 counter packets 0 bytes 0 # handle 10
    }
}
```

Figura 6.75 - Reglas ejemplo ct mark

5. [Equipo:PC1 - Usuario:root] Ahora, desde PC1 probamos a realizar ping a Encaminador y al DNS de Google (8.8.8.8) (ver Figura 6.76).

Equipo:PC1 Usuario:root

```
$ ping -c 3 10.10.1.10
$ ping -c 3 8.8.8.8
```

```
[root@localhost dit]# ping 10.10.1.10
PING 10.10.1.10 (10.10.1.10) 56(84) bytes of data.
64 bytes from 10.10.1.10: icmp_seq=1 ttl=64 time=0.623 ms
64 bytes from 10.10.1.10: icmp_seq=2 ttl=64 time=0.674 ms
64 bytes from 10.10.1.10: icmp_seq=3 ttl=64 time=0.761 ms
^C
--- 10.10.1.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 49ms
rtt min/avg/max/mdev = 0.623/0.686/0.761/0.056 ms
[root@localhost dit]# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 77ms
```

Figura 6.76 - Ping PC1 ejemplo ct mark

Como se puede ver, el ping realizado al equipo Encaminador se recibe, mientras que el de Google no recibe respuesta, pasaremos a comprobar ahora la tabla de conexiones en Encaminador.

6. [Equipo:Encaminador - Usuario:root] Se comprueban las entradas de conntrack en el equipo Encaminador.

```
Equipo:Encaminador Usuario:root
$ conntrack -L
```

Podemos ver en la Figura 6.77, como se ha creado la entrada para el flujo de comunicación ICMP con el Encaminador, además se puede ver también que este tiene como marca de conexión, la 2, que previamente había sido asignada en el *hook input* de la tabla filter.

Sin embargo, no se ve ninguna entrada del flujo de comunicación con el DNS de Google, esto es debido a que al ser marcado en el *hook forward* con la marca 0x3, en el *hook postrouting* se realiza el *drop* de los paquetes cuyo tráfico tenga la marca 0x3.

Al no conseguir atravesar el *hook postrouting*, conntrack no crea la entrada de este en su tabla.

```
root@localhost:/home/dit# conntrack -L
icmp      1 29 src=10.10.1.20 dst=10.10.1.10 type=8 code=0 id=11420 src=10.10.1.10 dst=10.10.1.20 type=0 code=0 id=11420 mark=2 use=1
conntrack v1.4.6 (conntrack-tools): 1 flow entries have been shown.
```

Figura 6.77 - Flujo de conexión ejemplo ct mark 0x2

Conclusión del ejemplo:

En este ejemplo se ha podido comprobar la potencia de una de las opciones más importantes del módulo conntrack.

La marca de conexión, permite que los administradores de red, puedan marcar los diferentes flujos de comunicación que atraviesan por el sistema de Netfilter, para posteriormente, darle un tratamiento específico a cada uno de ellos.

Esta marca, puede usarse para marcar aquellas comunicaciones que se consideren sospechosas, pudiendo así darles un tratamiento especial.

6.8.2 Ejemplo state

Descripción del ejemplo:

Para este ejemplo, se usará el escenario de 2 equipos “Anexo E - Escenario 2 Equipos”.

Partimos de la estructura de tablas y cadenas creada en el ejemplo anterior “6.8.1 Ejemplo mark”

- Se van a crear reglas en Encaminador que permitan los estados de conexión New y Established, que se realicen hacia el servidor Apache de la máquina, mientras que realizaremos el *drop* de conexiones salientes que tengan el estado New o Established.
- El equipo PC1, será el que realice la petición al servidor Apache de Encaminador y peticiones a la página de Google, para ver, que esta debe ser denegada.

Análisis del ejemplo:

1. [Equipo:Encaminador - Usuario:root] Se crean las siguientes reglas en nftables:
 - 1) Se realiza la limpieza de las reglas de la cadena input de la tabla filter.
 - 2) Se realiza la limpieza de las reglas de la cadena forward de la tabla filter.

- 3) Se realiza la limpieza de las reglas de la cadena postrouting de la tabla filter.
- 4) En la cadena forward de filter, se añade una regla que realice el *drop* de los paquetes que tengan Ip origen, la de PC1 y cuyo estado sea New o Established.
- 5) En la cadena input de filter, se añade una regla que realice el *accept* de los paquetes cuyo estado sea New o Established.

Equipo: Encaminador **Usuario:** root

```
$ nft flush chain ip filter input
$ nft flush chain ip filter forward
$ nft flush chain ip filter postrouting

$ nft add rule ip filter forward ip 'saddr 10.10.1.20/32 ct state { new, established } counter drop'
$ nft add rule ip filter input 'ct state { new, established } counter
```

2. [Equipo:Encaminador - Usuario:root] Se visualiza el *ruleset* de nftables de Encaminador:

En Encaminador, activamos también el programa Wireshark, para que escuche tráfico en la interfaz “ens38”

En la Figura 6.78, se puede comprobar la salida del *ruleset* de nftables.

Equipo: Encaminador **Usuario:** root

```
$ nft list ruleset -a
```

```
root@localhost:/home/dit# nft list ruleset -a
table ip nat {
    chain prerouting {
        type nat hook prerouting priority 0; policy accept;
        ip daddr 192.168.106.136 dnat to 10.10.1.20 # handle 6
    }

    chain postrouting {
        type nat hook postrouting priority 100; policy accept;
        ip saddr 10.10.1.20 counter packets 0 bytes 0 masquerade # handle 7
    }
}
table ip filter {
    chain prerouting {
        type filter hook prerouting priority 0; policy accept;
    }

    chain postrouting {
        type filter hook postrouting priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
    }

    chain input {
        type filter hook input priority 0; policy accept;
        ct state { established, new } counter packets 4067 bytes 13040864 accept # handle 7
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
        ip saddr 10.10.1.20 ct state { established, new } counter packets 396 bytes 32484 drop # handle 12
    }
}
root@localhost:/home/dit#
```

Figura 6.78 - Ruleset ejemplo ct state

3. **[Equipo:PC1 - Usuario:root]** A continuación desde PC1, usando el buscador Firefox, accedemos al servidor Apache de Encaminador (10.10.1.10:80)

4. **[Equipo:Encaminador - Usuario:root]** Se comprueba la salida de Wireshark y se muestra también las entradas de conntrack.

```
Equipo:Encaminador Usuario:root
```

```
$ conntrack -L
```

```
root@localhost:/home/dit# conntrack -L
tcp      6 431996 ESTABLISHED src=10.10.1.20 dst=10.10.1.10 sport=37620 dport=80 src=10.10.1.10 dst=10.10.1.20 sport=80 dport=37620 [ASSURED] mark=0 use=1
conntrack v1.4.6 (conntrack-tools): 1 flow entries have been shown.
```

Figura 6.79 - Flujo conexión servidor Apache

En la Figura 6.79, se puede ver que al mostrar las entradas de conntrack, aparece una única entrada. Esta entrada hace referencia a la conexión de PC1 con el servidor Apache de Encaminador.

En la Figura 6.80, se puede ver la comunicación de PC1 con el servidor de Encaminador, no solo los paquetes de apertura de conexión TCP, sino la petición de la página web que realiza PC1.

No.	Time	Source	Destination	Protocol	Length	Info
4	3.223266326	10.10.1.20	8.8.8.8	DNS	87	Standard query 0x701d AAAA services.addons.mozilla.org
5	5.006987928	10.10.1.20	8.8.8.8	DNS	107	Standard query 0x5f4f A content-signature-2.cdn.mozilla.net.localdomain
6	5.007013402	10.10.1.20	8.8.8.8	DNS	107	Standard query 0x9759 AAAA content-signature-2.cdn.mozilla.net.localdomain
7	5.960750480	10.10.1.20	10.10.1.10	TCP	74	37622 - 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1420955842 TSecr=0 WS=128
8	5.960819099	10.10.1.10	10.10.1.20	TCP	74	80 - 37622 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2888379711 TSecr=1420955842
9	5.960971771	10.10.1.20	10.10.1.10	TCP	66	37622 - 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1420955843 TSecr=2888379711
10	5.961779039	10.10.1.20	10.10.1.10	HTTP	505	GET / HTTP/1.1
11	5.961809746	10.10.1.20	10.10.1.10	TCP	66	80 - 37622 [ACK] Seq=1 Ack=440 Win=64768 Len=0 TSval=2888379712 TSecr=1420955844
12	5.962613940	10.10.1.20	10.10.1.10	HTTP	3543	HTTP/1.1 200 OK (text/html)
13	5.962774078	10.10.1.20	10.10.1.10	TCP	66	37622 - 80 [ACK] Seq=440 Ack=3478 Win=36224 Len=0 TSval=1420955845 TSecr=2888379713
14	6.161634763	10.10.1.20	10.10.1.10	HTTP	459	GET /icons/ubuntu-logo.png HTTP/1.1
15	6.161877489	10.10.1.10	10.10.1.20	HTTP	246	HTTP/1.1 304 Not Modified
16	6.162051847	10.10.1.20	10.10.1.10	TCP	66	37622 - 80 [ACK] Seq=833 Ack=3658 Win=39168 Len=0 TSval=1420956044 TSecr=2888379912
17	6.231471920	10.10.1.20	8.8.8.8	DNS	76	Standard query 0xabff A httpd.apache.org
18	6.231515299	10.10.1.20	8.8.8.8	DNS	76	Standard query 0xf704 AAAA httpd.apache.org
19	6.231583538	10.10.1.20	8.8.8.8	DNS	78	Standard query 0xf58a A buqs.launchpad.net

▶ Frame 7: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
 ▶ Ethernet II, Src: VMware_62:ed:ad (00:0c:29:62:ed:ad), Dst: VMware_f0:01:1e (00:0c:29:f0:01:1e)
 ▶ Internet Protocol Version 4, Src: 10.10.1.20, Dst: 10.10.1.10
 ▶ Transmission Control Protocol, Src Port: 37622, Dst Port: 80, Seq: 0, Len: 0
 Source Port: 37622
 Destination Port: 80
 [Stream index: 0]
 [TCP Segment Len: 0]
 Sequence number: 0 (relative sequence number)
 [Next sequence number: 0 (relative sequence number)]
 Acknowledgment number: 0
 1010 = Header Length: 40 bytes (10)
 Flags: 0x002 (SYN)
 Window size value: 29200
 [Calculated window size: 29200]
 Checksum: 0x8c6d [unverified]
 [Checksum Status: Unverified]
 ...
 0000 00 0c 29 f0 01 1e 00 0c 29 62 ed ad 00 00 45 00 ..)....b...E
 0010 00 3c bf 06 40 00 40 00 64 b4 0a 0a 01 14 0a 0a .< @@ d...
 0020 01 0a 92 f6 00 50 6b f2 ee a3 00 00 00 00 a0 02 ...Pk.
 0030 72 10 6c 6d 00 00 02 04 05 b4 04 02 08 0a 54 b2 r.lm....T
 0040 10 c2 00 00 00 00 01 03 03 07

Figura 6.80 - Captura Tráfico conexión PC1 servidor Apache de Encaminador

5. **[Equipo:PC1 - Usuario:root]** Ahora se probará a pedir desde PC1 la página web de Google.

```
Equipo:PC1 Usuario:root
```

```
$ wget google.es
```

En la Figura 6.81, se ve como se intenta realizar la obtención de la página www.google.es.

Viendo así, como no se puede resolver el nombre del dominio, tal y como se esperaba, al ser esta consulta un nuevo flujo de conexión.

```
[root@localhost dit]# wget www.google.es
--2021-01-19 01:46:49--  http://www.google.es/
Resolving www.google.es (www.google.es)... failed: Name or service not known.
wget: unable to resolve host address 'www.google.es'
[root@localhost dit]#
```

Figura 6.81 - Petición a Google desde PC1

6. [Equipo:Encaminador - Usuario:root] Se comprueba la salida de Wireshark y se muestra el *ruleset* de nftables, en el cual, se podrá ver el aumento del contador de la regla *drop* de forward.

No.	Time	Source	Destination	Protocol	Length: Info
1	0.000000000	10.10.1.20	8.8.8.8	DNS	73 Standard query 0xce33 A www.google.es
2	0.000058343	10.10.1.20	8.8.8.8	DNS	73 Standard query 0x0a3c AAAA www.google.es
3	5.000660502	10.10.1.20	8.8.8.8	DNS	73 Standard query 0xce33 A www.google.es
4	5.006105808	10.10.1.20	8.8.8.8	DNS	73 Standard query 0x0a3c AAAA www.google.es
5	5.335484768	Vmware_62:ed:ad	Vmware_f0:01:1e	ARP	60 Who has 10.10.1.10? Tell 10.10.1.20
6	5.335509737	Vmware_f0:01:1e	Vmware_62:ed:ad	ARP	42 10.10.1.10 is at 00:0c:29:f0:01:1e
7	10.011061594	10.10.1.20	8.8.8.8	DNS	85 Standard query 0xc866 A www.google.es.localdomain
8	10.011106939	10.10.1.20	8.8.8.8	DNS	85 Standard query 0xea85 AAAA www.google.es.localdomain
9	15.013387744	10.10.1.20	8.8.8.8	DNS	85 Standard query 0xc866 A www.google.es.localdomain
10	15.013431069	10.10.1.20	8.8.8.8	DNS	85 Standard query 0xea85 AAAA www.google.es.localdomain

Figura 6.82 - Captura tráfico sin resolución DNS PC1

Se puede observar en la Figura 6.82, que se realiza la captura de tráfico en la interfaz “ens38” de Encaminador y que a pesar de realizar peticiones de consulta al servidor DNS (Ip: 8.8.8.8), no vemos las respuestas de este.

También podemos ver como en la Figura 6.83, aumentan el número de paquetes a los que se les ha realizado el *drop* por la regla de estados de ct.

```
chain forward {
    type filter hook forward priority 0; policy accept;
    ip saddr 10.10.1.20 ct state { established, new } counter packets 912 bytes 73530 drop # handle 12
}

localhost:/home/dit#
```

Figura 6.83 - Aumento de paquetes drop en la cadena forward

Conclusión del ejemplo:

En este ejemplo se ha podido comprobar y afianzar el uso de los estados de los paquetes dentro del módulo conntrack.

Este punto es de especial interés para un administrador de redes, ya que en él, se entiende la base de los flujos de comunicación, como resumen se recuerda que:

- No es el primer paquete que ve conntrack, el que abre una entrada en este, sino que existen paquetes que son incapaces de abrir una entrada, simplemente pueden ser asociados a una entrada existente o su estado será inválido.
- El estado *related* actúa como el estado *new* de conexiones que dependen de un flujo de comunicación previo, como puede ser FTP.
- El punto de categorización de los estados, es en los *hooks prerouting* y *output*, en una fase temprana de procesado (prioridad -200).

- La expectación es el tipo de entradas que usa conntrack, para trabajar con flujos de comunicación *related*.
- Y por último, recordar los campos y los posibles valores que se pueden tener de las diferentes entradas de conntrack.

7 TRADUCCIÓN DE DIRECCIONES (NAT)

En este capítulo se analizará la traducción de direcciones, viendo como nftables realiza este proceso y en que *hooks* tiene lugar.

La traducción de direcciones [56], es un método que consiste en cambiar la dirección y/o el puerto (origen o destino).

Este método, es el que se usa en cada hogar para que los routers de las empresas puedan sacar nuestro tráfico a la red, sin el uso del mismo seríamos incapaces de comunicarnos.

El principal punto fuerte de la traducción de direcciones es, por ejemplo, permitir que el tráfico de una red privada vaya a otra red privada o al exterior, o bien, que el tráfico proveniente de una red pública pueda entrar dentro de una red privada.

7.1 Snat y Masquerade

Snat permite cambiar la dirección y o el puerto origen de un paquete, mientras que Masquerade, realiza la misma operación, pero cambiando la dirección origen por la dirección de la interfaz por la que sale el paquete.

Como se vio en el punto “Instalación Nftables”, al realizar la instalación el módulo NAT se carga por defecto, por lo que pasaremos directamente a realizar las pruebas.

7.1.1 Ejemplo Snat

Descripción del Ejemplo:

En este ejemplo se mostrará el uso de Snat, así como su funcionamiento.

Para ello se usará el escenario de 2 equipos “Anexo E - Escenario 2 Equipos”.

Se usará Snat, para permitir que el tráfico de PC1 sea capaz de salir a máquinas públicas, permitiendo así la conexión de este equipo con Internet.

Se usará además Wireshark en el equipo Encaminador, para monitorizar el tráfico de las interfaces “ens33” y “ens38”.

La dirección de la red interna es 10.10.1.0/24, de forma que Encaminador actúa como equipo intermedio que encamina tráfico de PC1 hacia el exterior.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Se activa Wireshark para que escuche en la interfaz “ens33” y en la interfaz “ens38”.
2. **[Equipo:PC1 - Usuario:root]** Se activa Wireshark para que escuche en la interfaz “ens38” y se intenta acceder desde PC1 a la web de trajano.

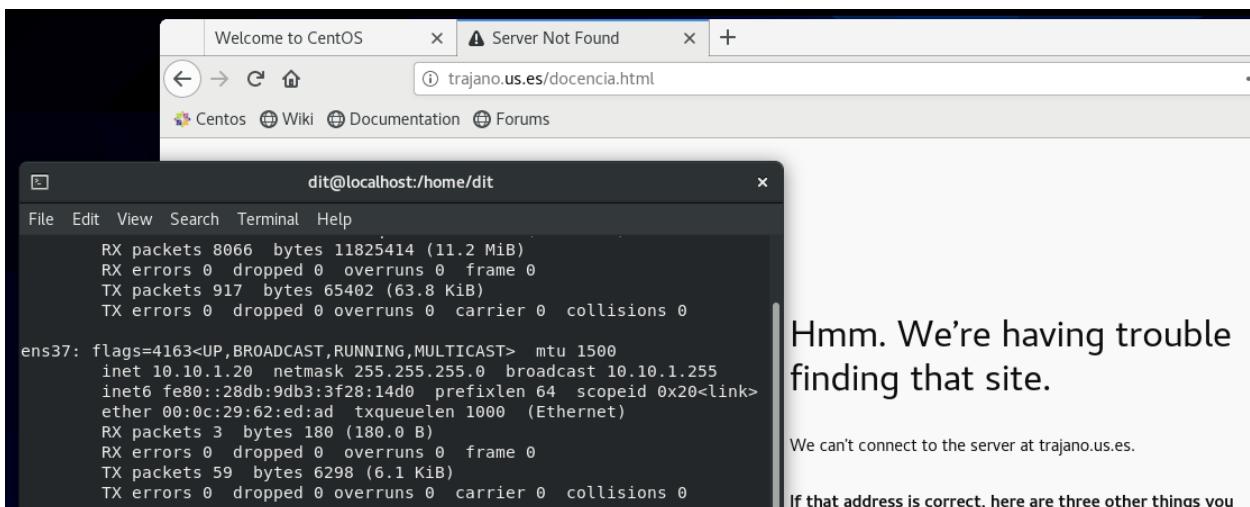


Figura 7.1 - Intento conexión trajano desde PC1 sin NAT

Como podemos ver en la Figura 7.1, no se puede acceder a la web, si ahora comprobamos las capturas de Wireshark tanto en “ens33” como en “ens38” veremos el porqué.

- [Equipo:Encaminador - Usuario:root] Se comprueba la captura realizada por Wireshark en la interfaz “ens38”.

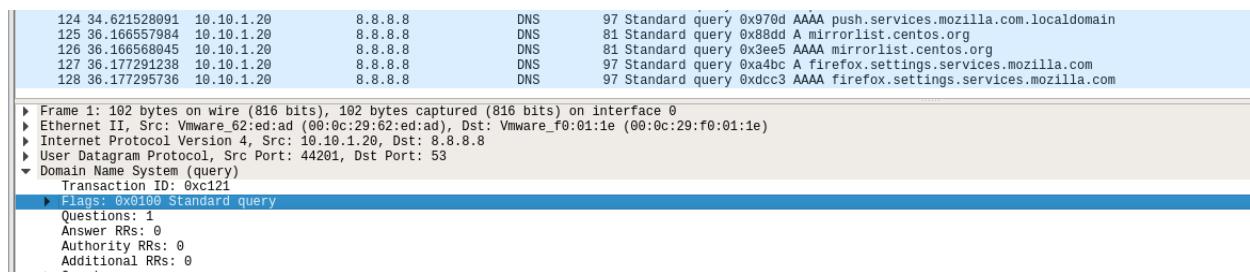


Figura 7.2 - Petición DNS lanzada por PC1 sin NAT en Encaminador

En la Figura 7.2, se ve que en la interfaz “ens38” está llegando la consulta DNS que está realizando PC1, para poder averiguar la Ip de la página de trajano.

Como Encaminador tiene activado el *forwarding*, este debe coger el paquete y encaminarlo hacia Internet.

- [Equipo:Encaminador - Usuario:root] Se comprueba la captura realizada por Wireshark en la interfaz “ens33”.

99	188.274398327	Vmware_f8:b6:0f	Broadcast	ARP	60 Who has 10.10.1.20? Tell 192.168.106.2
100	189.273917902	Vmware_f8:b6:0f	Broadcast	ARP	60 Who has 10.10.1.20? Tell 192.168.106.2
101	189.760395655	10.10.1.20	8.8.8.8	DNS	102 Standard query 0xce7b A incoming.telemetry.mozilla.org.localdomain
102	189.760427763	10.10.1.20	8.8.8.8	DNS	102 Standard query 0xaa87 AAAA incoming.telemetry.mozilla.org.localdomain
103	189.772512239	10.10.1.20	8.8.8.8	DNS	93 Standard query 0xdbc6 A mirrorlist.centos.org.localdomain
104	189.772543663	10.10.1.20	8.8.8.8	DNS	93 Standard query 0xc4d1 AAAA mirrorlist.centos.org.localdomain
105	190.274504261	Vmware_f8:b6:0f	Broadcast	ARP	60 Who has 10.10.1.20? Tell 192.168.106.2
106	191.275054517	Vmware_f8:b6:0f	Broadcast	ARP	60 Who has 10.10.1.20? Tell 192.168.106.2
107	191.660261041	10.10.1.20	8.8.8.8	DNS	96 Standard query 0x2087 A detectportal.firefox.com.localdomain
108	191.660292067	10.10.1.20	8.8.8.8	DNS	96 Standard query 0x3293 AAAA detectportal.firefox.com.localdomain
109	192.275409679	Vmware_f8:b6:0f	Broadcast	ARP	60 Who has 10.10.1.20? Tell 192.168.106.2

► Frame 105: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
 ► Ethernet II, Src: Vmware_f8:b6:0f (00:50:56:f8:b6:0f), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
 ▾ Address Resolution Protocol (request)
 Hardware type: Ethernet (1)
 Protocol type: IPv4 (0x0800)
 Hardware size: 6
 Protocol size: 4
 Opcode: request (1)
 Sender MAC address: Vmware_f8:b6:0f (00:50:56:f8:b6:0f)

Figura 7.3 - Petición DNS de PC1 sale por Encaminador sin NAT

En la Figura 7.3, se puede ver que la petición sale de Encaminador por la interfaz “ens33”, pero se puede ver también que hay un mensaje de difusión (Broadcast) de tipo ARP, preguntando por la Ip 10.10.1.20, y esta pregunta vemos que la lanza nuestro router con dirección 192.168.106.2.

Esta petición solo tiene validez local y no se encamina, por lo que al preguntar por esa Ip y al no tenerla Encaminador, no se puede encaminar de vuelta la respuesta del DNS hacia PC1.

Para subsanar este problema, aplicaremos la traducción de direcciones (NAT).

En concreto, usaremos Snat en el *hook postROUTING* que es el último *hook* por el que pasa el paquete antes de abandonar Encaminador.

5. **[Equipo:Encaminador - Usuario:root]** Se crea la siguiente estructura en nftables para poder aplicarle a los paquetes de PC1 la traducción de direcciones.
 - 1) Se crea la tabla nat
 - 2) Se crean dos cadenas dentro de esta tabla, una cadena prerouting con prioridad -100 y una cadena postrouting con prioridad 100.
 - 3) Se muestra el *ruleset* de reglas de nftables para comprobar que estas han sido creadas correctamente.

```
Equipo:Encaminador Usuario:root

# Creación de la tabla
$ nft add table ip nat

# Creación de las cadenas
$ nft add chain 'ip nat preroutingnat { type nat hook prerouting priority -100 ; }'
$ nft add chain 'ip nat postroutingnat { type nat hook postrouting priority 100 ; }'

# Mostrar el ruleset
$ nft list ruleset
```

```

root@localhost:/home/dit# nft list ruleset
table ip nat {
    chain preroutingnat {
        type nat hook prerouting priority 100; policy accept;
    }

    chain postroutingnat {
        type nat hook postrouting priority -100; policy accept;
    }
}
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
    }
}
root@localhost:/home/dit# █

```

Figura 7.4 - Armazón ejemplo Snat y Dnat

Como se ve en la estructura de la Figura 7.4, se ha creado una tabla llamada nat y dos cadenas de tipo nat, con los *hooks* *prerouting* y *postrouting*.

6. [Equipo:Encaminador - Usuario:root] Ahora se creará una regla de Snat en la cadena postrouting, de forma que cambiaremos el origen del paquete que realiza la petición DNS, para que la Ip origen sea la de la interfaz “ens33” de Encaminador.

Comprobamos que una vez creada esta regla, el *ruleset* de nftables queda tal y como se puede ver en la Figura 7.5.

Equipo:Encaminador Usuario:root

```
$ nft add rule ip nat postroutingnat ip saddr 10.10.1.20 ip daddr
8.8.8.8 counter snat to 192.168.106.136
```

```

root@localhost:/home/dit# nft list ruleset
table ip nat {
    chain preroutingnat {
        type nat hook prerouting priority 100; policy accept;
    }

    chain postroutingnat {
        type nat hook postrouting priority -100; policy accept;
        ip saddr 10.10.1.20 ip daddr 8.8.8.8 counter packets 2 bytes 174 snat to 192.168.106.136
    }
}
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
    }
}
root@localhost:/home/dit# █

```

Figura 7.5 - Regla Snat en Encaminador

7. [Equipo:PC1 - Usuario:root] Se vuelve a realizar la petición a la web de trajano desde PC1.

8. [Equipo:PC1 - Usuario:root] Se comprueba la salida de Wireshark de la interfaz “ens38” y “ens33”, del equipo Encaminador.

1 5.78898650 10.10.1.20 8.8.8.8	DNS	73 Standard query 0x2c17 A trajano.us.es
6 5.788974210 10.10.1.20 8.8.8.8	DNS	73 Standard query 0x8b1e AAAA trajano.us.es
7 5.835237571 8.8.8.8 10.10.1.20	DNS	89 Standard query response 0x2c17 A trajano.us.es A 193.147.162.130
8 5.856768317 8.8.8.8 10.10.1.20	DNS	120 Standard query response 0x8b1e AAAA trajano.us.es SOA onix.us.es
9 5.858326030 10.10.1.20 193.147.162.130	TCP	74 48162 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TStamp=1151950318 TSrc=0 WS=128
10 6.119068570 10.10.1.20 193.147.162.130	TCP	74 48164 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TStamp=1151950318 TSsrc=0 WS=128
Frame 5: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0		
Ethernet II, Src: Vmware_62:ed:ad (00:0c:29:62:ed:ad), Dst: Vmware_f0:01:1e (00:0c:29:f0:01:1e)		
Internet Protocol Version 4, Src: 10.10.1.20, Dst: 8.8.8.8		
0100 = Version: 4		
.... 0101 = Header Length: 20 bytes (5)		
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)		
Total Length: 59		
Identification: 0x22ef (8943)		
Flags: 0x4000, Don't fragment		
Time to live: 64		
Protocol: UDP (17)		
Header checksum: 0xfc95 [validation disabled]		
[Header checksum status: Unverified]		
Source: 10.10.1.20		
Destination: 8.8.8.8		
User Datagram Protocol, Src Port: 60987, Dst Port: 53		
Domain Name System (query)		
Transaction ID: 0x2c17		
Flags: 0x0100 Standard query		
Questions: 1		
Answer RRs: 0		
Authority RRs: 0		
Additional RRs: 0		
Queries		
[Response In: 7]		

Figura 7.6 - Paquete Req y Res DNS

Vemos en la Figura 7.6, como en la interfaz “ens38” ya capturamos la petición y la respuesta del DNS, se puede ver en el paquete señalado que tiene Id Ip (8943), siendo este la petición de 10.10.1.20 (PC1) al servidor DNS (8.8.8.8)

Si ahora nos fijamos en la Figura 7.7, vemos que el paquete marcado, tiene dirección origen 192.168.106.136 y destino el servidor DNS y que además el Id Ip de este paquete coincide con el visto en la figura anterior (8943).

Por lo que el equipo Encaminador, lo que ha realizado es una modificación en el paquete, cambiando su dirección de origen por la que se le ha indicado en la regla creada anteriormente.

15 19.006044288 192.168.106.136 8.8.8.8	DNS	73 Standard query 0x2c17 A trajano.us.es
16 19.006080562 192.168.106.136 8.8.8.8	DNS	73 Standard query 0x8b1e AAAA trajano.us.es
17 19.052300680 8.8.8.8 192.168.106.136	DNS	89 Standard query response 0x2c17 A trajano.us.es A 193.147.162.130
18 19.073774552 8.8.8.8 192.168.106.136	DNS	120 Standard query response 0x8b1e AAAA trajano.us.es SOA onix.us.es
Frame 15: 73 bytes on wire (584 bits), 73 bytes captured (584 bits) on interface 0		
Ethernet II, Src: Vmware_f0:01:14 (00:0c:29:f0:01:14), Dst: Vmware_f8:b6:0f (00:50:56:f8:b6:0f)		
Internet Protocol Version 4, Src: 192.168.106.136, Dst: 8.8.8.8		
0100 = Version: 4		
.... 0101 = Header Length: 20 bytes (5)		
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)		
Total Length: 59		
Identification: 0x22ef (8943)		
Flags: 0x4000, Don't fragment		
Time to live: 63		
Protocol: UDP (17)		
Header checksum: 0xdd82 [validation disabled]		
[Header checksum status: Unverified]		
Source: 192.168.106.136		
Destination: 8.8.8.8		
User Datagram Protocol, Src Port: 60987, Dst Port: 53		
Domain Name System (query)		

Figura 7.7 - Paquete Req y Res DNS con Snat aplicado en Encaminador

Cuando se aplica una regla de traducción de direcciones, el sistema guarda en memoria una tupla de correspondencia de dirección y puerto origen y la dirección y puerto destino, de forma que cuando llegue la respuesta al equipo, este sepa deshacer el cambio realizado, lo que se conoce como Snat inverso (aunque como veremos en el siguiente punto esto es un Dnat).

Si vamos a PC1, veremos que todavía no tenemos conexión con el servidor, ya que este lanza la petición TCP al servidor trajano y en las reglas, solo está permitido las peticiones al DNS.

9. [Equipo:Encaminador - Usuario:root] Se crea una regla que permita la comunicación con servidores web, para ello se habilita que el puerto destino sea el 80 o el 443(TLS).

Equipo:Encaminador Usuario:root

```
$ nft add rule ip' nat postROUTINGnat ip saddr 10.10.1.20/32 tcp
dport { 80, 443 } counter snat to 192.168.106.136'
$ nft list ruleset
```

10. [Equipo:PC1 - Usuario:root] Se vuelve a probar desde PC1 la conexión con el servidor web, comprobando que este ya se comunica con nuestro equipo, tal y como se puede comprobar en la Figura 7.8.

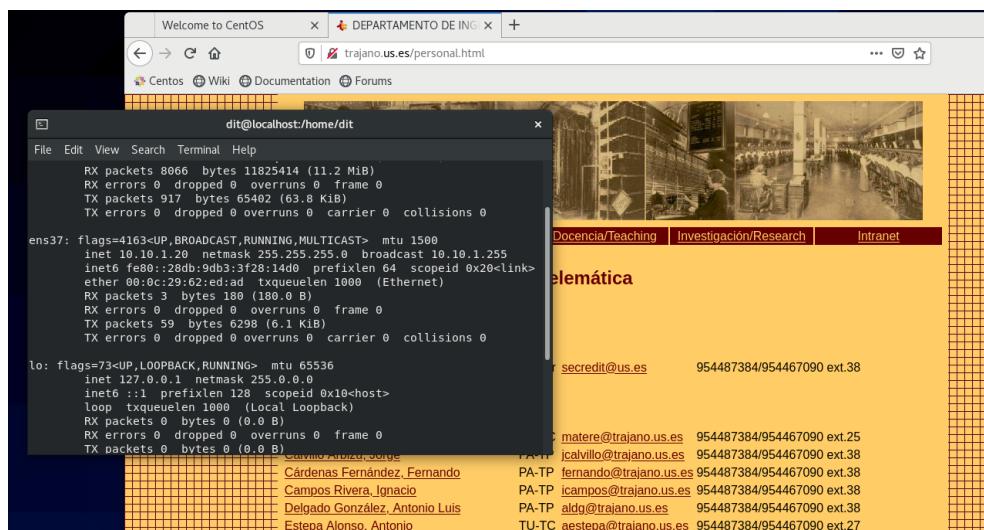


Figura 7.8 - Conexión con Servidor web Trajano

11. [Equipo:Encaminador - Usuario:root] Se comprueba la captura realizada por Wireshark en la interfaz “ens38” y “ens33” del equipo Encaminador.

No.	Time	Source	Destination	Protocol	Length	Info
155	15.28476970	10.10.1.20	193.147.162.130	TCP	74	48474 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1154534930 TSecr=0 WS=128
156	15.32686328	193.147.162.130	10.10.1.20	TCP	58	80 → 48474 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
157	15.32769430	10.10.1.20	193.147.162.130	TCP	60	48474 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
158	15.33320908	10.10.1.20	193.147.162.130	TCP	74	48476 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1154534979 TSecr=0 WS=128
159	15.37486276	193.147.162.130	10.10.1.20	TCP	58	80 → 48476 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
160	15.37502664	10.10.1.20	193.147.162.130	TCP	60	48476 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
161	20.33399865	10.10.1.20	193.147.162.130	TCP	60	48474 → 80 [FIN, ACK] Seq=1 Ack=1 Win=29200 Len=0
162	20.33473767	193.147.162.130	10.10.1.20	TCP	54	80 → 48474 [ACK] Seq=1 Ack=2 Win=64239 Len=0
163	20.37922532	193.147.162.130	10.10.1.20	TCP	54	80 → 48474 [FIN, PSH, ACK] Seq=1 Ack=2 Win=64239 Len=0
164	20.37850605	10.10.1.20	193.147.162.130	TCP	60	48474 → 80 [ACK1] Seq=2 Ack=2 Win=29200 Len=0
Frame 155: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0						
Ethernet II, Src: VMware_62:ed:ad (00:0c:29:62:ed:ad), Dst: VMware_f0:01:1e (00:0c:29:0f:01:1e)						
Internet Protocol Version 4, Src: 10.10.1.20, Dst: 193.147.162.130						
0100 = Version: 4						
.... 0101 = Header Length: 20 bytes (5)						
Differentiated Services Field: 0x00 (DS0: CS0, ECN: Not-ECT)						
Total Length: 60						
Identification: 0x10f1 (4337)						
Flags: 0x0000 (Not fragment)						
Time to live: 64						
Protocol: TCP (6)						
Header checksum: 0xb997 [validation disabled]						
[Header checksum status: Unverified]						
Source: 10.10.1.20						
Destination: 193.147.162.130						
Transmission Control Protocol, Src Port: 48474, Dst Port: 80, Seq: 0, Len: 0						
Source Port: 48474						
Destination Port: 80						
[Stream index: 7]						
[TCP Segment Len: 0]						
Sequence number: 0 (relative sequence number)						
[Next sequence number: 0 (relative sequence number)]						
Acknowledgment number: 0						
1010 = Header Length: 40 bytes (10)						
Flags: 0x002 (SYN)						

Figura 7.9 - Captura ens38 Encaminador conexión Trajano con Snat completo

Como se puede ver en la Figura 7.9, en la interfaz “ens38” de Encaminador, ya vemos la conexión con Trajano y se puede ver como conecta con el servidor y este le responde y le sirve la página web.

NO.	TIME	SOURCE	DESTINATION	Protocol	Length	INFO
29	13.030919028	192.168.106.136	193.147.162.130	TCP	74	48462 → 80 [SYN] Seq=0 Win=29200 MSS=1460 SACK_PERM=1 TStamp=1154527812 TSecr=0 WS=128
30	13.030993808	193.147.162.130	192.168.106.136	TCP	68	89 → 48462 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
31	13.07324343985	192.168.106.136	193.147.162.130	TCP	54	48467 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
32	13.733769725	192.168.106.136	193.147.162.130	HTTP	436	GET /personal.html HTTP/1.1
33	13.734028862	192.168.106.136	192.168.106.136	TCP	69	48467 → 80 [ACK] Seq=1 Ack=383 Win=64240 Len=0
34	13.859396882	193.147.162.130	192.168.106.136	HTTP	2364	HTTP/1.1 200 OK [text/html]
35	13.859594020	192.168.106.136	193.147.162.130	TCP	54	48462 → 80 [ACK] Seq=383 Ack=2311 Win=33580 Len=0
36	14.622283958	192.168.106.136	193.224.219.111	TCP	74	38518 → 443 [SYN] Seq=0 Win=20300 MSS=1460 SACK_PERM=1 TStamp=3150387531 TSecr=0 WS=128
37	14.625421615	192.168.106.136	8.8.8.8	DNS	95	Standard query 0x9c2c A content-signature-2.cdn.mozilla.net
38	14.625457933	192.168.106.136	8.8.8.8	DNS	95	Standard query 0x3d34 AAAA content-signature-2.cdn.mozilla.net
39	14.627224057	192.168.106.136	8.8.8.8	DNS	90	Standard query 0x6375 A incoming.telemetry.mozilla.org
40	14.627258928	192.168.106.136	8.8.8.8	DNS	90	Standard query 0xdd7c AAAA incoming.telemetry.mozilla.org
41	14.627456563	192.168.106.136	35.244.247.133	TCP	74	33522 → 443 [SYN] Seq=0 Win=29200 MSS=1460 SACK_PERM=1 TStamp=2704463916 TSecr=0 WS=128

► Frame 29: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
 ► Ethernet II, Src:Vmware_f0:01:14 (00:0c:29:f0:01:14), Dst:Vmware_f8:b6:0f (00:50:56:f8:b6:0f)
 ► Internet Protocol Version 4, Src: 192.168.106.136, Dst: 193.147.162.130
 ▾ Transmission Control Protocol, Src Port: 48462, Dst Port: 80, Seq: 0, Len: 0
 Source Port: 48462
 Destination Port: 80
 [Stream index: 1]
 [TCP Segment Len: 0]
 Sequence number: 0 (relative sequence number)
 [Next sequence number: 0 (relative sequence number)]
 Acknowledgment number: 0
 1010 ... = Header Length: 40 bytes (10)
 ► Flags: 0x002 (SYN)
 Window size value: 29200
 [Collected window size: 29200]
 Checksum: 0xb0ef (unverified)
 [Checksum Status: Unverified]
 Urgent pointer: 0
 ► Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
 [Timestamps]

Figura 7.10 - Captura ens33 Encaminador conexión Trajano Snat completo

En la Figura 7.10, se puede ver como Encaminador ha realizado el Snat a los paquetes procedentes de PC1, con destino a la web de Trajano (193.147.106.136).

12. **[Equipo:Encaminador - Usuario:root]** Si vemos las reglas de Encaminador en la Figura 7.11, se puede ver como ha aumentado el contador de las reglas de Snat tanto de DNS como de la conexión HTTP.

```
• root@localhost: /home/dit#
Archivo Editar Ver Buscar Terminal Ayuda
root@localhost:/home/dit# nft list ruleset -a
table ip nat {
    chain preroutingnat {
        type nat hook prerouting priority 100; policy accept;
    }

    chain postroutingnat {
        type nat hook postrouting priority -100; policy accept;
        ip saddr 10.10.1.20 ip daddr 8.8.8.8 counter packets 197 bytes 14563 snat to 192.168.106.136 # handle 5
        ip saddr 10.10.1.20 tcp dport { http, https } counter packets 38 bytes 2280 snat to 192.168.106.136 # handle 9
    }
}
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
    }
}
root@localhost:/home/dit#
```

Figura 7.11 - Reglas y contadores Snat de Encaminador

Conclusión del ejemplo:

Como se ha podido ver en el ejemplo, Snat permite cambiar el origen de los paquetes, para que estos puedan salir a redes distintas y puedan mantener la comunicación.

También se ha podido ver que, a la hora de hacer la traducción de direcciones, no podemos olvidarnos, que antes de la fase de comunicación con la página web, se debe consultar la dirección de esta en el

DNS, por lo que se deberá tener en cuenta realizarle la traducción de direcciones a este tipo de mensajes.

Una vez visto el ejemplo de Snat que es la base, veremos Masquerade [57] que es una traducción más específica de tipo Snat.

La diferencia entre uno y otro, es que el cambio que realiza Snat es estático, de forma que la Ip a la que se traduce es siempre la fijada en la regla, mientras que con Masquerade la Ip origen se sustituye por la Ip de la interfaz por la que sale el paquete.

De forma que, si se desea usar este tipo de reglas en sistemas dinámicos, en los cuales las direcciones Ip's las sirve un servidor DHCP de manera dinámica, será más conveniente usar reglas Masquerade antes que reglas Snat.

7.1.2 Ejemplo Masquerade

Descripción del ejemplo:

Se va a ver un ejemplo en el cual, se sustituirán las reglas Snat por Masquerade y comprobaremos que se puede seguir accediendo al servidor web, aún incluso cuando cambie la dirección que el DHCP asigna a la interfaz “ens33” de Encaminador.

Para realizar este ejemplo, partiremos de la estructura creada en el ejemplo anterior, la cual se basa en el “Anexo E - Escenario 2 Equipos”.

Análisis del ejemplo.

1. **[Equipo:Encaminador - Usuario:root]** Lo primero que se debe hacer, es borrar las reglas de Snat creadas anteriormente.

```
Equipo:Encaminador Usuario:root  
$ nft flush chain ip nat postROUTINGnat
```

2. **[Equipo:Encaminador - Usuario:root]** Ahora se procede a crear las mismas reglas, pero usando Masquerade en lugar de Snat.

Además, se activa Wireshark para que capture tráfico en las interfaces “ens33” y “ens38”.

```
Equipo:Encaminador Usuario:root  
$ nft add rule ip nat postROUTINGnat ip saddr 10.10.1.20/32 ip  
daddr 8.8.8.8 counter masquerade  
$ nft add rule ip' nat postROUTINGnat ip saddr 10.10.1.20/32 tcp  
dport { 80, 443 } counter masquerade'
```

3. **[Equipo:PC1 - Usuario:root]** Una vez creadas las reglas, probamos la conexión desde PC1 hacia el servidor Trajano.

4. **[Equipo:Encaminador - Usuario:root]** Se comprueban las capturas de tráfico de Wireshark.

En la Figura 7.12 y Figura 7.13, se puede observar cómo se realiza la petición al servidor y como se produce la traducción de direcciones (Masquerade), para comunicarse con este y que así nos sirva la página web.

No.	Time	Source	Destination	Protocol	Length	Info
1	6.4600000000000005	193.147.162.130	193.147.162.130	TCP	58 48552 - 88 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1158796284 TSeср=0 WS=128	
2	6.4600000000000005	193.147.162.130	193.147.162.130	TCP	58 80 - 48552 [SYN, ACK] Seq=1 Ack=1 Win=64240 Len=0 MSS=1460	
3	6.4600000000000005	193.147.162.130	193.147.162.130	TCP	60 48552 - 88 [ACK] Seq=1 Ack=1 Win=29200 Len=0	
4	6.4600000000000005	193.147.162.130	193.147.162.130	HTTP	433 GET /index.html HTTP/1.1	
5	6.014637705	193.147.162.130	193.147.162.130	TCP	54 80 - 48552 [ACK] Seq=388 Win=64240 Len=0	
6	6.015252860	193.147.162.130	193.147.162.130	DNS	73 Standard query 0x1cdd A trajano.us.es	
7	6.015321364	193.147.162.130	193.147.162.130	DNS	73 Standard query 0xdæ2 AAAA trajano.us.es	
8	6.014237995	193.147.162.130	193.147.162.130	HTTP	1864 HTTP/1.1 200 OK (text/html)	
9	6.014237995	193.147.162.130	193.147.162.130	TCP	60 48552 - 88 [ACK] Seq=388 Ack=1811 Win=32120 Len=0	
10	6.014237995	193.147.162.130	193.147.162.130	DNS	120 Standard query response 0x1cdd A trajano.us.es SOA onix.us.es	
11	6.014237995	193.147.162.130	193.147.162.130	DNS	69 Standard query response 0x1cdd A trajano.us.es A 193.147.162.130	
12	6.014237995	193.147.162.130	193.147.162.130	HTTP	360 GET /include/css/tematica.css HTTP/1.1	
13	6.014237995	193.147.162.130	193.147.162.130	TCP	54 80 - 48552 [ACK] Seq=1811 Ack=686 Win=64240 Len=0	
14	6.014237995	193.147.162.130	193.147.162.130	TCP	60 48552 - 88 [FIN, ACK] Seq=686 Ack=1811 Win=32120 Len=0	
15	6.014237995	193.147.162.130	193.147.162.130	TCP	54 80 - 48552 [ACK] Seq=1811 Ack=687 Win=64239 Len=0	
16	6.014237995	193.147.162.130	193.147.162.130	HTTP	879 HTTP/1.1 200 OK (text/css)	
▶ Frame 4: 433 bytes on wire (3464 bits), 433 bytes captured (3464 bits) on interface 0						
▶ Ethernet II, Src: VMware_62:ed:ad (00:0c:29:f0:01:1e), Dst: VMware_f0:01:1e (00:0c:29:f0:01:1e)						
▶ Internet Protocol Version 4, Src: 10.10.1.20, Dst: 193.147.162.130						
0100 .. 0101 = Version: 4						
.. 0101 = Header Length: 20 bytes (5)						
▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)						
Total Length: 419						
Identification: 0xc64b (50763)						
▶ Flags: 0x4000, Don't fragment						
Time to live: 64						
Protocol: TCP (6)						
Header checksum: 0x03d6 [validation disabled]						
[Header checksum status: Unverified]						
Source: 10.10.1.20						
Destination: 193.147.162.130						

Figura 7.12 - Captura ens38 Encaminador usando Masquerade

No.	Time	Source	Destination	Protocol	Length	Info
4	7.221940022	192.168.106.136	193.147.162.130	TCP	74 48552 - 88 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1158796284 TSeср=0 WS=128	
5	7.249652745	193.147.162.130	192.168.106.136	TCP	60 80 - 48552 [SYN, ACK] Seq=1 Ack=1 Win=64240 Len=0 MSS=1460	
6	7.2500027379	192.168.106.136	193.147.162.130	TCP	54 48552 - 88 [ACK] Seq=1 Ack=1 Win=29200 Len=0	
7	9.236499352	192.168.106.136	193.147.162.130	TCP	493 Standard query 0x1cdd A trajano.us.es	
8	9.236499352	193.147.162.130	192.168.106.136	TCP	60 80 - 48552 [ACK] Seq=1 Ack=380 Win=64240 Len=0	
9	9.237187453	192.168.106.136	192.168.106.136	DNS	73 Standard query 0x1cdd A trajano.us.es	
10	9.237187453	192.168.106.136	192.168.106.136	DNS	73 Standard query 0xdæ2 AAAA trajano.us.es	
11	9.264012738	193.147.162.130	192.168.106.136	HTTP	1864 HTTP/1.1 200 OK (text/html)	
12	9.264273194	192.168.106.136	193.147.162.130	TCP	54 4855 - 88 [ACK] Seq=388 Ack=1811 Win=32120 Len=0	
13	9.275213762	192.168.106.136	192.168.106.136	DNS	120 Standard query response 0xdæ2 AAAA trajano.us.es SOA onix.us.es	
14	9.275213762	192.168.106.136	193.147.162.130	DNS	69 Standard query response 0x1cdd A trajano.us.es A 193.147.162.130	
15	10.051834532	192.168.106.136	193.147.162.130	HTTP	360 GET /include/css/tematica.css HTTP/1.1	
16	10.052029345	193.147.162.130	192.168.106.136	TCP	60 80 - 48552 [ACK] Seq=1811 Ack=686 Win=64240 Len=0	
▶ Frame 1: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0						
▶ Ethernet II, Src: VMware_f0:01:14 (00:0c:29:f0:01:14), Dst: VMware_f0:b6:0f (00:0c:56:f8:b6:0f)						
▶ Internet Protocol Version 4, Src: 10.10.1.14, Dst: 10.10.1.14						
▶ Transmission Control Protocol, Src Port: 34590, Dst Port: 443, Seq: 1, Ack: 1, Len: 0						
Source Port: 34590						
Destination Port: 443						
[Stream index: 0]						
[TCP Segment Len: 0]						
Sequence number: 1 (relative sequence number)						
Next sequence number: 1 (relative sequence number)						
Acknowledgment number: 1 (relative ack number)						
Offset = Header Length: 20 bytes (5)						
▶ Flags: 0x010 (ACK)						
Window size value: 39420						
[Calculated window size: 39420]						
[Window size scaling factor: -1 (unknown)]						
Checksum: 0x00d4 [unverified]						
[Checksum Status: Unverified]						
Urgent pointer: 0						
▶ [Timestamps]						

Figura 7.13 - Captura ens33 Encaminador usando Masquerade

5. **[Equipo:Encaminador - Usuario:root]** Ahora, se reinicia el enlace para que nuestro router nos dé por DHCP otra dirección para la interfaz “ens33”.

En la Figura 7.14, se puede observar la nueva dirección Ip que nos sirve el servidor DHCP.

Equipo:Encaminador Usuario:root						
<pre>\$ ip link set ens33 down \$ ip link set ens33 up \$ ifconfig</pre>						

```

[1] Archivo Editar Ver Buscar Terminal Ayuda
● root@localhost: /home/dit
Archivo Editar Ver Buscar Terminal Ayuda
RX packets 220 bytes 13676 (13.6 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 220 bytes 13676 (13.6 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@localhost:/home/dit# ifconfig
root@localhost:/home/dit# ifconfig
ens33: flags=4163 mtu 1500
        inet 192.168.106.158 netmask 255.255.255.0 broadcast 192.168.106.255
                inet6 fe80::10a:34ff:fe42:9508 prefixlen 64 scopeid 0x20<link>
                  ether 00:0c:29:f0:01:14 txqueuelen 1000 (Ethernet)
                    RX packets 11687 bytes 863237 (8.6 MB)
                    RX errors 0 dropped 0 overruns 0 frame 0
                    TX packets 6734 bytes 61930 (61.9 KB)
                    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens38: flags=4163 mtu 1500
        inet 10.10.1.10 netmask 255.255.255.0 broadcast 10.10.1.255
                inet6 fe80::20c:29ff:fe01:1e txqueuelen 64 scopeid 0x20<link>
                  ether 00:0c:29:f0:01:1e txqueuelen 1000 (Ethernet)
                    RX packets 5498 bytes 531831 (531.8 KB)
                    RX errors 0 dropped 0 overruns 0 frame 0
                    TX packets 2069 bytes 1669872 (1.0 MB)
                    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens39: flags=4163 mtu 1500
        inet 172.168.0.10 netmask 255.255.255.0 broadcast 172.168.0.255
                inet6 fe80::20c:29ff:fe01:28 txqueuelen 1000 (Ethernet)
                  ether 00:0c:29:f0:01:28 txqueuelen 1000 (Ethernet)
                    RX packets 0 bytes 0 (0.0 B)
                    RX errors 0 dropped 0 overruns 0 frame 0
                    TX packets 101 bytes 12818 (12.8 KB)
                    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

```

● root@localhost: /home/dit
Archivo Editar Ver Buscar Terminal Ayuda
root@localhost:/home/dit# nft list ruleset
table ip nat {
    chain preroutingnat {
        type nat hook prerouting priority 100; policy accept;
    }
    chain postroutingnat {
        type nat hook postrouting priority -100; policy accept;
        ip saddr 10.10.1.20 ip daddr 8.8.8.8 counter packets 22 bytes 1409 masquerade
        ip saddr 10.10.1.20 tcp dport { http, https } counter packets 9 bytes 540 masquerade
    }
}
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }
    chain forward {
        type filter hook forward priority 0; policy accept;
    }
    chain output {
        type filter hook output priority 0; policy accept;
    }
}
root@localhost:/home/dit#

```

Figura 7.14 - Nueva dirección Ip ens33 con Masquerade

Una vez que el router nos ha dado una nueva dirección, probamos a volver a pedir la página web desde PC1 y veremos como la regla sigue funcionando sin necesidad de que la actualicemos, como hubiese sido necesario si hubiéramos usado Snat.

6. [Equipo:PC1 - Usuario:root] Volvemos a probar la conexión desde PC1 hacia el servidor Trajano.
7. [Equipo:Encaminador - Usuario:root] Se comprueban las capturas de tráfico de Wireshark, comprobando que se tiene conexión y que la regla Masquerade, se ha encargado de cambiar la dirección origen del paquete por la nueva dirección que tiene la interfaz “ens33”.

La Figura 7.15, es la captura de Wireshark en la interfaz “ens38”, en la cual puede verse como se completa la conexión con el servidor Trajano.

En la Figura 7.16, se puede observar como Masquerade, realiza la traducción de direcciones con la nueva dirección de la interfaz “ens33”.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.10.1.20	8.8.8.8	DNS	73	Standard query 0xbbed A trajano.us.es
2	0.048894174	8.8.8.8	10.10.1.20	DNS	89	Standard query response 0xbbed A trajano.us.es A 193.147.162.130
3	0.066209937	10.10.1.20	193.147.162.130	TCP	74	89 Standard query response 0xbbed A trajano.us.es A 193.147.162.130 Seq=0 Win=29200 Len=0 MSS=1460 SACK Perm=1 TSval=1160584351 TSecr=0 WS=128
4	0.106808474	193.147.162.130	10.10.1.20	TCP	58	80 - 48560 [SYN, ACK] Seq=9 Ack=1 Win=64249 Len=0 MSS=1460
5	0.107011040	10.10.1.20	193.147.162.130	TCP	60	48560 - 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
6	3.605134773	10.10.1.20	8.8.8.8	DNS	73	Standard query 0x5d5c A trajano.us.es
7	3.6065211932	10.10.1.20	8.8.8.8	DNS	73	Standard query 0xe963 AAAA trajano.us.es
8	3.653081988	8.8.8.8	10.10.1.20	DNS	89	Standard query response 0x5d5c A trajano.us.es A 193.147.162.130
9	3.662238672	8.8.8.8	10.10.1.20	DNS	120	Standard query response 0xe963 AAAA trajano.us.es SOA onix.us.es
10	4.553984334	10.10.1.20	8.8.8.8	DNS	78	Standard query 0x48ed A departamento.us.es
11	4.553174820	10.10.1.20	8.8.8.8	DNS	78	Standard query 0xb1f6 AAAA departamento.us.es
12	4.631939222	8.8.8.8	10.10.1.20	DNS	116	Standard query response 0x48ed A departamento.us.es CNAME mvaloja.us.es A 193.147.175.173

Figura 7.15 - Captura ens38 Encaminador con Masquerade nueva IP

No.	Time	Source	Destination	Protocol	Length Info
1	0.000000000	192.168.106.158	8.8.8.8	DNS	73 Standard query 0xbbed A trajano.us.es
2	0.048778928	192.168.106.158	192.168.106.158	DNS	89 Standard query response 0xbbed A trajano.us.es A 193.147.162.130
3	0.066294119	192.168.106.158	193.147.162.130	TCP	74 48560 - 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK PERM=1 TSeq=1160584351 TSecr=0 WS=128
4	0.106694113	193.147.162.130	192.168.106.158	TCP	60 80 - 48560 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
5	0.106961158	192.168.106.158	193.147.162.130	TCP	54 48566 - 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
6	3.605107577	192.168.106.158	8.8.8.8	DNS	73 Standard query 0x5d5c A trajano.us.es
7	3.605149481	192.168.106.158	8.8.8.8	DNS	73 Standard query 0xe963 AAAA trajano.us.es
8	3.652970964	8.8.8.8	192.168.106.158	DNS	89 Standard query response 0x5d5c A trajano.us.es A 193.147.162.130
9	3.662277853	8.8.8.8	192.168.106.158	DNS	120 Standard query response 0xe963 AAAA trajano.us.es SOA onix.us.es

Frame 3: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
Ethernet II, Src: VMware_f0:01:14 (00:0c:29:f0:01:14), Dst: VMware_f8:b6:0f (00:50:56:f8:b6:0f)
Internet Protocol Version 4, Src: 192.168.106.158, Dst: 193.147.162.130
0100 ... = Version: 4
... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 60
Identification: 0x278a (10122)
Flags: 0x4000, Don't fragment
Time to live: 63
Protocol: TCP (6)
Header checksum: 0x84d5 [validation disabled]
[Header checksum status: Unverified]
Source: 192.168.106.158
Destination: 193.147.162.130
Transmission Control Protocol, Src Port: 48560, Dst Port: 80, Seq: 0, Len: 0

Figura 7.16 - Captura ens33 Encaminador con Masquerade nueva IP

En la Figura 7.17, se puede ver como se ha actualizado los contadores de las reglas creadas en nftables.

```
● root@localhost: /home/dit
Archivo Editar Ver Buscar Terminal Ayuda
root@localhost:/home/dit# nft list ruleset
table ip nat {
    chain preroutingnat {
        type nat hook prerouting priority 100; policy accept;
    }

    chain postroutingnat {
        type nat hook postrouting priority -100; policy accept;
        ip saddr 10.10.1.20 ip daddr 8.8.8.8 counter packets 50 bytes 3287 masquerade
        ip saddr 10.10.1.20 tcp dport { http, https } counter packets 43 bytes 2440 masquerade
    }
}
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
    }
}
```

Figura 7.17 - Reglas y contadores Masquerade de Encaminador

Conclusión del ejemplo:

Como se puede ver en las Figuras anteriores, PC1 ha conseguido conectarse con el servidor de Trajano y Encaminador ha conseguido realizar la traducción de direcciones, sin necesidad de cambiar las reglas, a pesar de que ha cambiado la Ip de la interfaz “ens33”.

Si esto se hubiera realizado con reglas Snat, la situación hubiera sido la misma que al comienzo de este capítulo, en el cual, el router seguiría preguntando por ARP la dirección antigua de la interfaz “ens33” de Encaminador, en lugar de preguntar por la nueva.

7.2 Dnat y Redirect

Al igual, que en el punto anterior, se realizaba la traducción de direcciones orígenes, en este punto se realizará la traducción de direcciones destino.

Se realizará el estudio de Redirect, el cual, es un caso específico de Dnat, en el que la dirección destino del paquete pasa a ser la de nuestro equipo.

Podemos hacer redirect con nftables, pero para poder crear una regla cuya acción terminal sea redirect, nuestra regla tiene que usar el módulo TCP o UDP.

7.2.1 Ejemplo Redirect

Descripción del ejemplo:

Para realizar este ejemplo usaremos el escenario de 3 equipos, “Anexo F - Escenario 3 Equipos”.

- PC2 será el encargado de generar tráfico desde un rango de direccionamiento público.
- PC1 levantará su servidor Apache para aceptar conexiones en el puerto 80.
- Encaminador aplicará las reglas de nftables y hará el *forwarding* de los paquetes.

Para probar redirect, vamos a hacer que PC2 realice una petición al servidor Apache de PC1(10.10.1.20:80) y Encaminador aplicará la regla redirect para redirigirlo a su servidor Apache.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Se crea la siguiente estructura en nftables.
 - 1) Se crea la tabla ip nat
 - 2) Se crean dos cadenas dentro de esta tabla, prerouting con prioridad -100 y postrouting con prioridad 100.
 - 3) Se crea una regla dentro de prerouting que realice el redirect de todos los paquetes TCP con destino PC1 y puerto destino el 80.

Equipo:Encaminador Usuario:root

```
#Se crea la tabla
$ nft add table ip nat

#Se añaden las cadenas
$ nft add chain 'ip nat preroutingnat { type nat hook prerouting
priority -100 ; }'
$ nft add chain 'ip nat postroutingnat { type nat hook postrouting
priority 100 ; }'

#Se añade la regla de redirect
$ nft add rule ip nat preroutingnat ip daddr 10.10.1.20 tcp dport
80 counter redirect to 80
```

2. **[Equipo:Encaminador - Usuario:root]** Se activa Wireshark para que monitorice el tráfico en la interfaz “ens39”.
3. **[Equipo:PC2 - Usuario:root]** Desde Firefox, en el PC2, se intenta acceder a la dirección Ip 10.10.1.20:80.

No.	Time	Source	Destination	Protocol	Length	Info
4	537.562911515	172.168.0.20	10.10.1.20	TCP	74	57106 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1860800646 TSecr=1309409861
5	537.563118128	10.10.1.20	172.168.0.20	TCP	74	80 → 57106 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1309409861 TSecr=1309409861
6	537.523112352	172.168.0.20	10.10.1.20	TCP	66	57106 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1860800674 TSecr=1309408921
7	537.563694663	172.168.0.20	10.10.1.20	HTTP	387	GET / HTTP/1.1
8	537.563872892	10.10.1.20	172.168.0.20	TCP	66	80 → 57106 [ACK] Seq=1 Ack=322 Win=64896 Len=0 TSval=1309408982 TSecr=1860800702
9	537.690995429	10.10.1.20	172.168.0.20	TCP	2962	80 → 57106 [ACK] Seq=1 Ack=322 Win=64896 Len=2896 TSval=1309409109 TSecr=18608006
10	537.691284698	10.10.1.20	172.168.0.20	HTTP	647	HTTP/1.1 200 OK (text/html)
11	537.710722470	172.168.0.20	10.10.1.20	TCP	66	57106 → 80 [ACK] Seq=322 Ack=3478 Win=36224 Len=0 TSval=1860800861 TSecr=1309409861
12	542.694227830	10.10.1.20	172.168.0.20	TCP	66	80 → 57106 [FIN, ACK] Seq=3478 Ack=322 Win=64896 Len=0 TSval=1309414112 TSecr=1860800861
13	542.695227055	172.168.0.20	10.10.1.20	TCP	66	57106 → 80 [ACK] Seq=322 Ack=3479 Win=36224 Len=0 TSval=1860805846 TSecr=1309408921
14	542.695276967	10.10.1.20	172.168.0.20	TCP	66	80 → 57106 [ACK] Seq=3479 Ack=323 Win=64896 Len=0 TSval=1309414113 TSecr=18608006
15	542.739668767	Vmware_91:4b:f2	Vmware_f0:01:28	ARP	60	Who has 172.168.0.10? Tell 172.168.0.20
16	542.739692832	Vmware_91:4b:f2	Vmware_f0:01:28	ARP	42	172.168.0.10 is at 00:0c:29:f0:01:28
17	552.713167140	172.168.0.20	10.10.1.20	TCP	74	57108 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1860815864 TSecr=1309424131
18	552.713278480	10.10.1.20	172.168.0.20	TCP	74	80 → 57108 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1309424131
19	552.746886170	172.168.0.20	10.10.1.20	TCP	66	57108 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1860815864 TSecr=1309424131
20	552.746927253	172.168.0.20	10.10.1.20	HTTP	347	GET /icons/ubuntu-logo.png HTTP/1.1
21	552.746955366	10.10.1.20	172.168.0.20	TCP	66	80 → 57108 [ACK] Seq=1 Ack=282 Win=64896 Len=0 TSval=1309424165 TSecr=1860815866
22	553.234278856	10.10.1.20	172.168.0.20	TCP	2962	80 → 57108 [ACK] Seq=1 Ack=282 Win=64896 Len=0 TSval=1309424652 TSecr=1860815866
23	553.234412066	10.10.1.20	172.168.0.20	HTTP	794	HTTP/1.1 200 OK (PNG)
24	553.247297677	172.168.0.20	10.10.1.20	TCP	66	57108 → 80 [ACK] Seq=282 Ack=3625 Win=36480 Len=0 TSval=1860816385 TSecr=1309424131
25	553.919222190	172.168.0.20	10.10.1.20	HTTP	308	favicon.ico HTTP/1.1
26	553.919257524	10.10.1.20	172.168.0.20	TCP	66	80 → 57108 [ACK] Seq=3625 Ack=524 Win=64768 Len=0 TSval=1309425337 TSecr=1860815866
27	553.947000201	10.10.1.20	172.168.0.20	HTTP	554	HTTP/1.1 404 Not Found (text/html)
28	553.947298272	172.168.0.20	10.10.1.20	TCP	66	57108 → 80 [ACK] Seq=524 Ack=4113 Win=39424 Len=0 TSval=1860817098 TSecr=1309425337

▶ Transmission Control Protocol, Src Port: 80, Dst Port: 57106, Seq: 2897, Ack: 322, Len: 581
 ▶ [2 Reassembled TCP Segments (347 bytes): #9(2896), #10(581)]
 ▶ Hypertext Transfer Protocol
 ▶ HTTP/1.1 200 OK\r\n
 Date: Thu, 21 Jan 2021 22:41:54 GMT\r\n
 Server: Apache/2.4.29 (Ubuntu)\r\n
 Last-Modified: Fri, 15 Jan 2021 12:34:47 GMT\r\n
 ETag: "2aa6-5b8ef9908a307-gzip"\r\n
 Accept-Ranges: bytes\r\n
 Vary: Accept-Encoding\r\n
 Content-Encoding: gzip\r\n

Figura 7.18 - Redirect de servidor Apache al de la máquina Encaminador

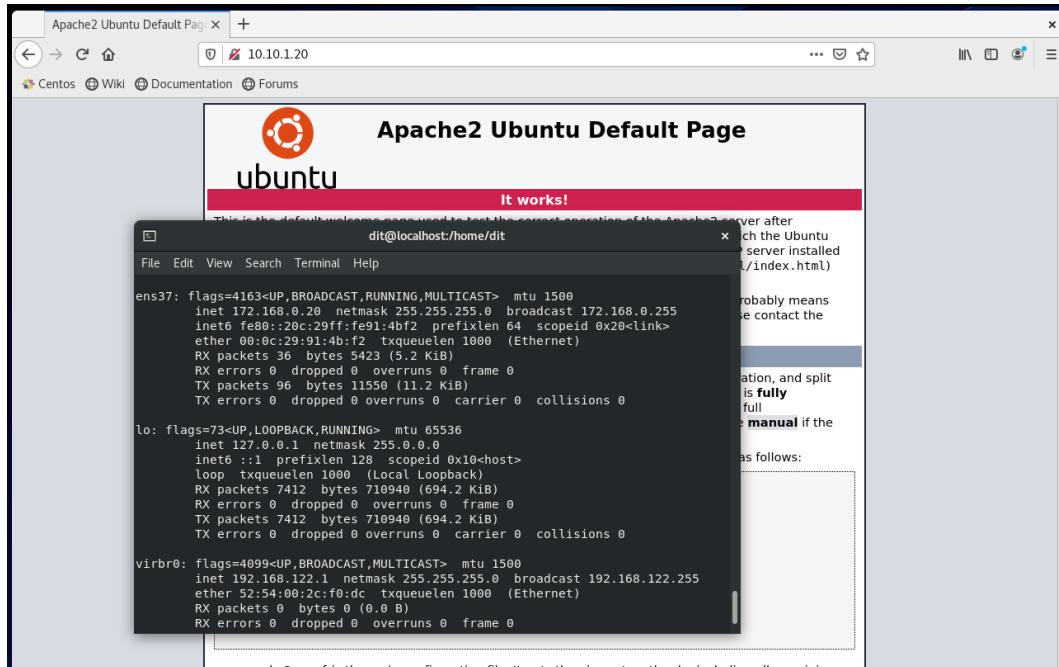


Figura 7.19 - Servidor Apache servido a PC2 con redirect en Encaminador

Como podemos ver en las Figura 7.18 y Figura 7.19, el servidor que ha recibido PC2 es el de la máquina Encaminador (10.10.1.10:80), a pesar de que la petición la ha realizado al servidor Apache de PC1 (10.10.1.20:80)

Podemos usar también redirect, para ocultar puertos al exterior, por ejemplo, se puede crear una regla que realice el *drop* del paquete si su puerto TCP destino es el 80, pero que si la petición tiene como puerto destino el 452 nos haga un redirect al 80 y nos sirva la página de Apache de Encaminador.

4. **[Equipo:Encaminador - Usuario:root]** Para probar como el Redirect puede ocultar un servicio se eliminarán las reglas de la cadena prerouting nat y se añadirán 2 nuevas reglas.

- La primera regla, se encarga de realizar el *drop* de los paquetes TCP que vayan dirigidos al puerto 80.
- La segunda regla, se encarga de realizar el redirect de puertos, de forma que todas las peticiones que se realicen al puerto 452 se estarán haciendo hacia el puerto 80 de la máquina.

Equipo: Encaminador **Usuario:** root

```
# Se borran las reglas de la cadena dada
$ nft flush chain ip nat preroutingnat

#Se añaden dos reglas para ocultar el servicio Apache
$ nft add rule ip nat preroutingnat tcp dport 80 counter drop
$ nft add rule ip nat preroutingnat tcp dport 452 counter redirect
to 80
```

5. [Equipo:PC2 - Usuario:root] Ahora realizamos la primera petición desde PC2 a la dirección (10.10.1.10:80) y como se puede ver en la Figura 7.20, no llega la respuesta del servidor y se realiza la retransmisión de los paquetes TCP.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	f800:20::20ff:fe00.. ff02::fb		MDNS	107	Standard query 0x0000 PTR _ipp._tcp.local, "OM" question PTR _ipp._tcp.local, "OM" question
2	0.000293014	172.168.0.10	224.0.0.251	MDNS	87	Standard query 0x0000 PTR _ipp._tcp.local, "OM" question PTR _ipp._tcp.local, "OM" question
3	45.879828789	172.168.0.20	10.10.1.10	TCP	74	48980 - 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=247751866 TSecr=0 WS=128
4	46.149834333	172.168.0.20	10.10.1.10	TCP	74	48982 - 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=247752137 TSecr=0 WS=128
5	46.899447864	172.168.0.20	10.10.1.10	TCP	74	[TCP Retransmission] 48980 - 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=247752875
6	47.207822292	172.168.0.20	10.10.1.10	TCP	74	[TCP Retransmission] 48982 - 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=247753195
7	48.935856801	172.168.0.20	10.10.1.10	TCP	74	[TCP Retransmission] 48980 - 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=247754923
8	49.055939715	172.168.0.20	10.10.1.10	TCP	74	[TCP Retransmission] 48982 - 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=247755243
9	51.367450885	Vmware_91:4b:f2	Vmware_f0:01:28	ARP	60	Who has 172.168.0.10? Tell 172.168.0.20
10	51.367466077	Vmware_f0:01:28	Vmware_91:4b:f2	ARP	42	172.168.0.10 is at 00:0c:29:f0:01:28
11	52.977247584	172.168.0.20	10.10.1.10	TCP	74	[TCP Retransmission] 48980 - 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=247758955
12	53.288063590	172.168.0.20	10.10.1.10	TCP	74	[TCP Retransmission] 48982 - 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=247759275
13	61.095777350	172.168.0.20	10.10.1.10	TCP	74	[TCP Retransmission] 48980 - 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=247767083
14	61.607942561	172.168.0.20	10.10.1.10	TCP	74	[TCP Retransmission] 48982 - 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=247767595
15	77.486581505	172.168.0.20	10.10.1.10	TCP	74	[TCP Retransmission] 48980 - 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=247783467

Figura 7.20 - Petición Apache de Encaminador puerto 80 con redirect de puerto

6. [Equipo:PC2 - Usuario:root] Ahora probamos a realizar la petición desde PC2 a la dirección (10.10.1.10:452), y como vemos en la Figura 7.21, la petición se ha realizado a ese puerto y es desde la dirección de Encaminador y desde el puerto 452 desde el cuál se sirve la página web (Figura 7.22).

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.168.0.20	10.10.1.10	TCP	74	51370 -> 452 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=247984366 TSecr=0 WS=128
2	0.000141121	10.10.1.10	172.168.0.20	TCP	74	452 -> 51370 [SYN ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=1313569229
3	0.000381995	172.168.0.20	10.10.1.10	TCP	66	51370 -> 452 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=247984397 TSecr=1313569229
4	0.000592028	172.168.0.20	10.10.1.10	HTTP	391	GET / HTTP/1.1
5	0.000655227	10.10.1.10	172.168.0.20	TCP	66	452 -> 51370 [ACK] Seq=1 Ack=326 Win=64896 Len=0 TSval=1313569230 TSecr=247984397
6	0.0022263998	10.10.1.10	172.168.0.20	HTTP	3543	HTTP/1.1 200 OK (text/html)
						Frame 2: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
						Ethernet II, Src: Vmware_f0:01:28 (00:0c:29:f0:01:28), Dst: Vmware_91:4b:f2 (00:0c:29:91:4b:f2)
						Internet Protocol Version 4, Src: 10.10.1.10, Dst: 172.168.0.20
						Transmission Control Protocol, Src Port: 452, Dst Port: 51370, Seq: 0, Ack: 1, Len: 0
						Source Port: 452
						Destination Port: 51370
						[Stream index: 0]
						[TCP Segment Len: 0]
						Sequence number: 0 (relative sequence number)
						[Next sequence number: 0 (relative sequence number)]
						Acknowledgment number: 1 (relative ack number)
						1010 = Header Length: 40 bytes (10)
						Flags: 0x012 (SYN, ACK)
						Window size value: 65160
						[Calculated window size: 65160]
						Checksum: 0xb7fe [unverified]
						Urgent pointer: 0
						Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
						[SEQ/ACK analysis]
						[Timestamps]

Figura 7.21 - Petición Apache de Encaminador puerto 452 con redirect de puerto

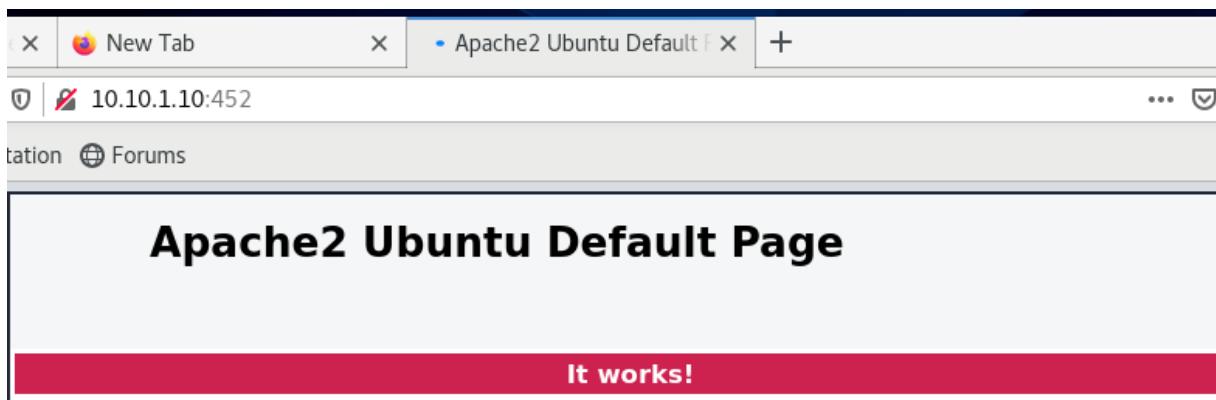


Figura 7.22 - Página web servida desde el puerto 452 de Encaminador con Redirect

Conclusión del ejemplo:

En este ejemplo, se ha podido ver el uso que tiene redirect.

Para un administrador de redes, puede resultar interesante usar esta opción para, ante una misma petición , realizada desde orígenes distintos, ofrecer servicios distintos.

De forma que , al igual que se ha realizado en el ejemplo, la página del servidor Apache de Encaminador, solo sea visible si la petición se realiza desde una red interna, mientras que para el resto de peticiones se mostrará el servidor Apache de la red externa.

7.3 El módulo conntrack en NAT

En el capítulo anterior “Seguimiento de Conexiones(Conntrack)”, se ha podido ver cómo funciona el módulo conntrack, como funciona su tabla de conexiones y los diferentes estados que esta puede tomar.

En este punto, se verá como el módulo NAT necesita del módulo conntrack para poder realizar sus traducciones de direcciones de forma Stateful.

Antes de comenzar, se debe ver la diferencia que existe entre los dos modelos de NAT que pueden implementarse con nftables.

El primer tipo de NAT que se verá es el Stateless NAT.

7.3.1 Stateless NAT

El Stateless NAT [58], se caracteriza por no realizar el seguimiento de la conexión, el cual se le aplica a la traducción de direcciones.

Este tipo de NAT, se lleva a cabo con una capacidad que se implementaba en iptables y que ha sido heredada por nftables (mangle).

Con *mangle* podemos cambiar el valor de los datos de los campos del paquete.

Este tipo de NAT, no se hace en cadenas de tipo nat, como se comentó en la explicación de las cadenas de nftables, si no que al ser una transformación que se realiza sin el seguimiento de conexiones, se realiza en las cadenas prerouting y postrouting de tipo filter.

Además de crear la regla para modificar el campo del paquete, se debe añadir la sentencia *notrack* al final de la regla, para que el módulo conntrack sepa que no debe realizar el seguimiento de esa conexión, ya que podría

provocar una corrupción de las tablas de conexiones.

7.3.1.1 Ejemplo Stateless NAT

Descripción del ejemplo:

Para realizar el ejemplo de Stateless NAT se usará el escenario de 2 equipos, “Anexo E - Escenario 2 Equipos”. En él, se colocarán las reglas necesarias para hacer que el ping del equipo PC1 pueda salir al exterior y que la respuesta llegue hasta el equipo, sin necesidad de usar el seguimiento de conexiones y sin que los paquetes pasen por las cadenas de tipo nat.

Análisis del ejemplo:

1. [Equipo:Encaminador - Usuario:root] En Encaminador se crea la siguiente estructura de nftables.
 - 1) Se crea una tabla filter de tipo inet
 - Se crean las siguientes cadenas: prerouting, input, forward, output y postrouting.
 - 2) Se crea una tabla filteringress de tipo netdev
 - Se crea la cadena ens33, la cual procesará el tráfico de la interfaz “ens33”.
 - Se crea la cadena ens38, la cual procesará el tráfico de la interfaz “ens38”.

Equipo:Encaminador Usuario:root

```
#Se crea la tabla filter
$ nft add table inet filter

#Se añaden las cadenas a la tabla filter
$ nft add chain inet filter postrouting '{ type filter hook
postrouting priority 0 ; }'
$ nft add chain inet filter prerouting '{ type filter hook
prerouting priority 0 ; }'
$ nft add chain inet filter forward '{ type filter hook forward
priority 0 ; }'
$ nft add chain inet filter input '{ type filter hook input
priority 0 ; }'
$ nft add chain inet filter output '{ type filter hook output
priority 0 ; }'

#Se crea la tabla filteringress
$ nft add table netdev filteringress

#Se añaden las cadenas a la tabla filteringress
$ nft add chain netdev' filteringress ens33 { type filter hook
ingress device ens33 priority 0 ; }
$ nft add chain netdev' filteringress ens38 { type filter hook
ingress device ens38 priority 0 ; }
```

2. [Equipo:Encaminador - Usuario:root] Se añaden las reglas a las cadenas de filteringress, para que sean capaces de marcar el tráfico y poder realizar su seguimiento usando el monitor *nftrace*.

Equipo:Encaminador **Usuario:**root

```
$ nft add rule netdev filteringress ens33 ip protocol icmp counter  
nftrace set 1  
$ nft add rule netdev filteringress ens38 ip protocol icmp counter  
nftrace set 1
```

3. [Equipo:Encaminador - Usuario:root] A continuación, se crean las reglas del Stateless NAT, las cuales consistirán en lo siguiente:

- La primera regla, se aplicará en el *hook postrouting* a aquellos paquetes con dirección origen PC1 y se cambiará el valor de este campo por la Ip de la interfaz “ens33”.
- La segunda regla, se aplicará en el *hook prerouting* a aquellos paquetes con dirección origen la Ip de la interfaz “ens33” de Encaminador y se cambiará el valor del campo de la dirección destino por el de PC1.

Ambas reglas llevan la opción de *notrack*, para que no se realice el seguimiento de las conexiones.

Equipo:Encaminador **Usuario:**root

```
$ nft add rule inet filter postrouting ip protocol icmp ip saddr  
10.10.1.20 ip saddr set 192.168.106.137 counter notrack  
$ nft add rule inet filter prerouting ip protocol icmp daddr  
192.168.106.137 ip daddr set 10.10.1.20 counter notrack
```

La primera regla, se encarga de que el ping que envía PC1 salga con la dirección origen que se ha indicado en la regla.

Mientras que la segunda, se usa para que una vez llegue la respuesta y que, ante la falta de entradas en el sistema de seguimiento de conexiones, que nos permita saber a quién pertenecía el paquete, podamos deshacer el cambio realizado en postrouting y poner la dirección destino del ICMP como la Ip de PC1.

Esta configuración tiene un problema y es que inutiliza la comunicación de Encaminador, ya que le estamos diciendo que todos los paquetes cuya dirección destino sea, la Ip de la interfaz “ens33” de Encaminador, los cambie para que su dirección destino sea PC1.

4. [Equipo:Encaminador - Usuario:root] Se activa el monitor de *nftrace*.

Equipo:Encaminaor **Usuario:** root

```
$ nft monitor trace
```

5. [Equipo:PC1 - Usuario:dit] Se lanza el ping desde PC1 hacia la dirección de máquina pública 8.8.8.8

Equipo:PC1 **Usuario:** dit

```
$ ping -c 1 8.8.8.8
```

6. [Equipo:Encaminador - Usuario:root] Se comprueba la salida de monitor para ver el seguimiento del paquete.

```
root@localhost:/home/dit# nft monitor trace
trace id 85df3910 netdev filteringress ens38 packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:le ip saddr 10.10.1.20 ip daddr 8.8.8.8 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 45469 ip length 84 icmp type echo-request icmp code 0 icmp id 9193 icmp sequence 1
trace id 85df3910 netdev filteringress ens38 rule ip protocol icmp counter packets 0 bytes 0 nftra ce set 1 (verdict continue)
trace id 85df3910 netdev filteringress ens38 verdict continue
trace id 85df3910 netdev filteringress ens38
trace id 85df3910 inet filter prerouting verdict continue
trace id 85df3910 inet filter prerouting
trace id 85df3910 inet filter forward verdict continue
trace id 85df3910 inet filter forward
trace id 85df3910 inet filter postrouting packet: oif "ens33" ip saddr 192.168.106.137 ip daddr 8.8.8.8 ip dscp cs0 ip ecn not-ect ip ttl 63 ip id 45469 ip length 84 icmp type echo-request icmp co de 0 icmp id 9193 icmp sequence 1
trace id 85df3910 inet filter postrouting rule ip protocol icmp ip saddr 10.10.1.20 ip saddr set 1 92.168.106.137 counter packets 2 bytes 168 notrack (verdict continue)
trace id 85df3910 inet filter postrouting verdict continue
trace id 85df3910 inet filter postrouting
```

Figura 7.23 - Camino Icmp-request Stateless NAT

En la Figura 7.23, se puede observar el camino que sigue el paquete ICMP que llega por la interfaz “ens38” del equipo Encaminador.

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena ens38 de la tabla filteringress, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en la cadena ens38 de filteringress es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 4) En prerouting se procesa y se toma como veredicto *continue*.
- 5) Se consulta el encaminamiento del paquete, determinando que este ha de ser encaminado hacia la máquina pública con Ip 8.8.8.8.
- 6) Al tener que ser encaminado, el paquete continúa su camino por el *hook forward*, siendo aquí su veredicto de *continue*.
- 7) El paquete llega al *hook postrouting*, donde hace *match* con la primera regla, realizando el cambio del campo Ip origen del paquete.
- 8) Se toma como veredicto *continue* y el paquete sale del equipo Encaminador hacia la máquina pública.

Se puede comprobar además, que el ID ICMP del paquete, coincide en ambas figuras (Figura 7.23 y Figura 7.24), siendo este (9193).

También se puede observar, que la respuesta llega a la cadena prerouting, en la cual hace *match* con la regla creada para deshacer el cambio y que la dirección destino pase a ser 10.10.1.20.

```

trace id bb12b7ed netdev filteringress ens33 rule ip protocol icmp counter packets 0 bytes 0 nftrace set 1 (verdict continue)
trace id bb12b7ed netdev filteringress ens33 verdict continue
trace id bb12b7ed netdev filteringress ens33
trace id bb12b7ed inet filter prerouting packet: iif "ens33" ether saddr 00:50:56:f8:b6:0f ether daddr 00:0c:29:f0:01:14 ip saddr 8.8.8.8 ip daddr 10.10.1.20 ip dsctp cs0 ip ecn not-ect ip ttl 128 ip id 38022 ip length 84 icmp type echo-reply icmp code 0 icmp id 9193 icmp sequence 1 trace id bb12b7ed inet filter prerouting rule ip protocol icmp ip daddr 192.168.106.137 ip daddr set 10.10.1.20 counter packets 2 bytes 168 notrack (verdict continue)
trace id bb12b7ed inet filter prerouting verdict continue
trace id bb12b7ed inet filter prerouting
trace id bb12b7ed inet filter forward verdict continue
trace id bb12b7ed inet filter forward
trace id bb12b7ed inet filter postrouting verdict continue
trace id bb12b7ed inet filter postrouting

```

Figura 7.24 - Camino Icmp-reply Stateless NAT

En la Figura 7.24, se puede ver como la respuesta del ICMP anterior llega por la interfaz “ens33” que es aquella que da acceso a internet.

- 1) El paquete de respuesta sale de la máquina pública y llega a Encaminador por la interfaz “ens33”
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena ens38 de la tabla filteringress, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en la cadena ens33 de filteringress es *continue*, por lo que el paquete continua su camino hacia la cadena prerouting de filter.
- 4) El paquete llega al *hook prerouting*, donde hace *match* con la primera regla, realizando el cambio del campo Ip destino del paquete por el valor de la Ip de PC1, su veredicto es *continue*.
- 5) Se consulta el encaminamiento del paquete, determinando que este ha de ser encaminado hacia el equipo PC1.
- 6) Al tener que ser encaminado, el paquete continua su camino por el *hook forward*, siendo aquí su veredicto de *continue*.
- 7) El paquete llega al *hook postrouting*, donde su veredicto es *continue*, saliendo así por la interfaz “ens38” camino a PC1.

7. [Equipo:Encaminador - Usuario:root] Se comprueba el *ruleset* de nftables.

Equipo:Encaminador **Usuario:**root

```
$ nft list ruleset
```

Se puede observar en la Figura 7.25, las reglas creadas y como el contador ha aumentado 1 paquete, tanto en la regla de prerouting como la de postrouting.

También se puede observar que, si se desea imprimir las entradas de conntrack o bien observar si se producen eventos de creación de entradas, no existe ninguna entrada , ni se ha producido ningún evento (Figura 7.26).

```

table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
    }

    chain prerouting {
        type filter hook prerouting priority 0; policy accept;
        ip protocol icmp ip daddr 192.168.106.137 ip daddr set 10.10.1.2
0 counter packets 1 bytes 84 notrack
    }

    chain postrouting {
        type filter hook postrouting priority 0; policy accept;
        ip protocol icmp ip saddr 10.10.1.20 ip saddr set 192.168.106.13
7 counter packets 1 bytes 84 notrack
    }
}

```

Figura 7.25 - Reglas y contadores Stateless NAT

```

root@localhost:/home/dit# conntrack -L
conntrack v1.4.6 (conntrack-tools): 0 flow entries have been shown.
root@localhost:/home/dit# conntrack -E --event-mask ALL

```

Figura 7.26 - Carencia de tablas conntrack Stateless NAT

Conclusión del ejemplo:

Como se ha podido comprobar en el ejemplo, el método para realizar Stateless NAT, consiste simplemente en la modificación del campo correspondiente (IP origen - Snat, IP destino - Dnat) con el uso de set.

Se desaconseja su uso, puesto que es complicado discernir que paquetes pertenecen a comunicaciones del equipo Encaminador y que paquetes pertenecen a comunicaciones del equipo PC1.

El módulo de seguimiento de conexiones, no lanza ningún evento ni crea entradas, gracias a la opción usada de *notrack*.

Una vez visto este tipo de NAT, se procederá a ver el segundo tipo que se puede usar con nftables, el Stateful NAT.

7.3.2 Stateful NAT

El Stateful NAT [59], a diferencia del Stateless NAT, si se apoya en el uso de las entradas de la tabla de conexión para realizar la traducción de direcciones, asociándolas a un flujo de comunicación.

En este punto, se ha de dejar claro, que las entradas de NAT y las entradas de conntrack son las mismas, es decir, ambas usan las entradas creadas y mantenidas por conntrack.

Por lo cual, la entrada que se puede ver de NAT, es como las entradas analizadas en el capítulo de conntrack, con la diferencia de que al hablar del paquete esperado en la respuesta, veremos que al campo correspondiente se le habrá aplicado la traducción de direcciones.

Para este tipo de NAT se usan las acciones terminales, vistas al comienzo del capítulo (Snat, Dnat, Masquerade...)

Descripción del ejemplo:

Para realizar el ejemplo de Stateful NAT, se usará también el escenario de 2 equipos “Anexo E - Escenario 2 Equipos”

Se realizará la misma prueba que en el ejemplo de Stateless NAT, pero usando Masquerade en la cadena de postrouting y viendo que sin reglas adicionales, este es capaz de deshacer el cambio al paquete de respuesta que llegue al equipo Encaminador.

En este caso y a diferencia del ejemplo Stateless, las reglas de NAT deben crearse en una cadena específica para ello.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Partiendo del esquema del ejemplo anterior, “Ejemplo Stateless NAT”, se crea la tabla nat y las cadenas prerouting y postrouting de tipo nat.

- Se usa la prioridad -10 en la cadena de prerouting, en el caso de dejarle la prioridad por defecto (0), esta cadena sería cursada después de la cadena prerouting de filter.
- Del mismo modo, a postrouting se le da una prioridad de 10, para asegurarnos de que es la última cadena que se visita antes de la salida del paquete del equipo.
- Por último, se crea la regla en postrouting de la tabla nat, de forma que si llega un paquete ICMP cuya dirección origen es 10.10.1.20, se le realice el Masquerade para que pueda salir al exterior.

Equipo:Encaminador Usuario:root

```
$ nft add table ip nat

$ nft add chain ip nat prerouting '{ type nat hook prerouting
priority -10 ; }'
$ nft add chain ip nat postrouting '{ type nat hook postrouting
priority 10 ; }'

$ nft add rule ip nat postrouting ip protocol icmp ip saddr
10.10.1.20 counter masquerade
```

2. **[Equipo:Encaminador - Usuario:root]** Se activa la monitorización de las reglas y de los eventos de conntrack .

Equipo:Encaminador Usuario:root

```
# Terminal 1
$ nft monitor trace

# Terminal 2
$ conntrack -E --any-nat
```

3. **[Equipo:PC1 - Usuario:dit]** Desde PC1, se realiza el ping hacia la dirección Ip de Google (8.8.8.8).

Equipo:PC1 Usuario:dit

```
$ ping -c 1 8.8.8.8
```

4. [Equipo:Encaminador - Usuario:root] Se comprueba la salida del monitor de *nfttrace* y del monitor de conntrack.

En la Figura 7.27, se puede ver que el paquete entra por la interfaz “ens38”, que su dirección origen es la 10.10.1.20 y que la dirección destino es la 8.8.8.8, mientras que en la respuesta, se puede ver que entra por la interfaz “ens33”, que su dirección origen es la 8.8.8.8 mientras que la dirección destino es la 192.168.106.137, por lo que se ha realizado el Masquerade correctamente.

```
root@localhost:/home/dit# nft monitor trace
trace id 11b8b6f7 netdev filteringress ens38 packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad eth
er daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 8.8.8.8 ip dscp cs0 ip ecn not-ect ip ttl
64 ip id 29840 ip length 84 icmp type echo-request icmp code 0 icmp id 10682 icmp sequence 1
trace id 11b8b6f7 netdev filteringress ens38 rule ip protocol icmp counter packets 1 bytes 84 nftr
ace set 1 (verdict continue)
trace id 11b8b6f7 netdev filteringress ens38 verdict continue
trace id 11b8b6f7 netdev filteringress ens38
trace id 11b8b6f7 ip nat prerouting verdict continue
trace id 11b8b6f7 ip nat prerouting
trace id 11b8b6f7 inet filter prerouting verdict continue
trace id 11b8b6f7 inet filter prerouting
trace id 11b8b6f7 inet filter forward verdict continue
trace id 11b8b6f7 inet filter forward
trace id 11b8b6f7 inet filter postrouting verdict continue
trace id 11b8b6f7 inet filter postrouting
trace id 11b8b6f7 ip nat postrouting packet: oif "ens33" ip saddr 10.10.1.20 ip daddr 8.8.8.8 ip d
scp cs0 ip ecn not-ect ip ttl 63 ip id 29840 ip length 84 icmp type echo-request icmp code 0 icmp
id 10682 icmp sequence 1
trace id 11b8b6f7 ip nat postrouting rule ip protocol icmp ip saddr 10.10.1.20 counter packets 0 b
ytes 0 masquerade (verdict accept)
trace id 8a34c427 netdev filteringress ens33 packet: iif "ens33" ether saddr 00:50:56:f8:b6:0f eth
er daddr 00:0c:29:f0:01:14 ip saddr 8.8.8.8 ip daddr 192.168.106.137 ip dscp cs0 ip ecn not-ect ip
ttl 128 ip id 38173 ip length 84 icmp type echo-reply icmp code 0 icmp id 10682 icmp sequence 1
trace id 8a34c427 netdev filteringress ens33 rule ip protocol icmp counter packets 1 bytes 84 nftr
ace set 1 (verdict continue)
trace id 8a34c427 netdev filteringress ens33 verdict continue
trace id 8a34c427 netdev filteringress ens33
trace id 8a34c427 inet filter prerouting verdict continue
trace id 8a34c427 inet filter prerouting
trace id 8a34c427 inet filter forward verdict continue
trace id 8a34c427 inet filter forward
trace id 8a34c427 inet filter postrouting verdict continue
trace id 8a34c427 inet filter postrouting
```

Figura 7.27 - Camino masquerade Stateful NAT

En la Figura 7.27, se puede ver el camino tanto del request como del reply ICMP.

Camino del request:

- 1) El paquete ICMP sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena ens38 de la tabla filteringress, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en la cadena ens38 de filteringress es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 4) El paquete llega al *hook prerouting*, donde su veredicto es *continue*.
- 5) Se consulta el encaminamiento del paquete, determinando que este ha de ser encaminado hacia la máquina pública con Ip 8.8.8.8.
- 6) Al tener que ser encaminado, el paquete continúa su camino por el *hook forward*, siendo aquí su veredicto de *continue*.
- 7) El paquete llega al *hook postrouting*, donde hace *match* con la primera regla, realizando el Masquerade de la dirección origen.
- 8) Al cruzar por el *hook postrouting*, se crea la entrada en conntrack.

Camino del reply:

- 1) El paquete de respuesta sale de la máquina pública y llega a Encaminador por la interfaz “ens33”
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena ens33 de la tabla filteringress, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en la cadena ens33 de filteringress es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 4) El paquete llega al *hook prerouting*, donde se deshace el cambio de masquerade y su veredicto es *continue*.
- 5) Se consulta el encaminamiento del paquete, determinando que este ha de ser encaminado hacia el equipo PC1.
- 6) Al tener que ser encaminado, el paquete continúa su camino por el *hook forward*, siendo aquí su veredicto de *continue*.
- 7) El paquete llega el *hook postrouting*, donde su veredicto es *continue*, creando así la segunda entrada en conntrack, esta entrada ya nos indica que se ha visto tráfico en ambos sentidos de la comunicación.

A continuación, se comprueba en la Figura 7.28, que la monitorización de cualquier entrada que registre NAT , muestra dos entradas, una entrada de categoría NEW, en la que se puede ver como la expectación del paquete de respuesta ha aplicado la regla NAT, y otra entrada de UPDATE en la cual se ha visto tráfico de vuelta y se ha quitado la flag de UNREPLIED de la entrada.

```
root@localhost:/home/dit# conntrack -E --any-nat
[NEW] icmp 1 30 src=10.10.1.20 dst=8.8.8.8 type=8 code=0 id=10682 [UNREP
LIED] src=8.8.8.8 dst=192.168.106.137 type=0 code=0 id=10682
[UPDATE] icmp 1 30 src=10.10.1.20 dst=8.8.8.8 type=8 code=0 id=10682 src=8.
8.8.8 dst=192.168.106.137 type=0 code=0 id=10682
^Cconntrack v1.4.6 (conntrack-tools): 2 flow events have been shown.
root@localhost:/home/dit#
```

Figura 7.28 - Entrada conntrack NAT

Se ha visualizado el código de la entrada de la aplicación y el cambio inverso de la traducción NAT que se realiza en el *hook prerouting*, se puede realizar una comprobación sencilla de este punto.

Como se ha explicado anteriormente, el primer *hook* que ve el paquete nada más ser servido por el Driver de la NIC es ingress, por lo que, si el cambio inverso se realiza antes de prerouting, este *hook*, no debería ver el paquete con dirección origen 8.8.8.8 y dirección destino 192.168.106.137, mientras que ingress si debe ver este paquete, puesto que todavía no se le habría aplicado el cambio inverso.

5. **[Equipo:Encaminador - Usuario:root]** Para realizar esta prueba, se parte de la base anterior y se añaden las siguientes reglas.
 - La primera regla, buscará la coincidencia de Ip origen 8.8.8.8 e Ip destino 192.168.106.137 en la cadena ens33.
 - La segunda regla, buscará la misma coincidencia en la cadena de prerouting de la tabla filter.

Equipo: Encaminador **Usuario:** root

```
$ nft add rule netdev filteringress ens33 ip daddr 192.168.106.137  
ip saddr 8.8.8.8 counter
```

```
$ nft add rule inet filter prerouting ip daddr 192.168.106.137 ip  
saddr 8.8.8.8 counter
```

6. [Equipo:PC1 - Usuario:dit] Se vuelve a realizar el ping desde PC1 con dirección Ip destino 8.8.8.8.

7. [Equipo:Encaminador - Usuario:root] Se comprueba la salida del monitor de *ntrace*.

En las Figura 7.29 y Figura 7.30, se puede comprobar que efectivamente el cambio inverso se realiza antes del *hook prerouting*.

En la primera de ellas, se puede comprobar que en la cadena prerouting de filter, no se hace *match* con la regla creada, mientras que en la segunda figura, se puede ver que el contador de paquetes de ingress ha aumentado en 1 mientras que el de prerouting se mantiene a 0.

```
trace id 4e7c95d8 netdev filteringress ens33 packet: iif "ens33" ether saddr 00:50:56:f8:b6:0f eth  
er daddr 00:0c:29:f0:01:14 ip saddr 8.8.8.8 ip daddr 192.168.106.137 ip dscp cs0 ip ecn not-ect ip  
ttl 128 ip id 38187 ip length 84 icmp type echo-reply icmp code 0 icmp id 11523 icmp sequence 1  
trace id 4e7c95d8 netdev filteringress ens33 rule ip protocol icmp counter packets 6 bytes 504 nft  
race set 1 (verdict continue)  
trace id 4e7c95d8 netdev filteringress ens33 rule ip daddr 192.168.106.137 counter packets 2 bytes  
168 (verdict continue)  
trace id 4e7c95d8 netdev filteringress ens33 rule ip saddr 8.8.8.8 ip daddr 192.168.106.137 coun  
ter packets 1 bytes 84 (verdict continue)  
trace id 4e7c95d8 netdev filteringress ens33 verdict continue  
trace id 4e7c95d8 netdev filteringress ens33  
trace id 4e7c95d8 inet filter prerouting verdict continue  
trace id 4e7c95d8 inet filter prerouting  
trace id 4e7c95d8 inet filter forward verdict continue  
trace id 4e7c95d8 inet filter forward  
trace id 4e7c95d8 inet filter postrouting verdict continue  
trace id 4e7c95d8 inet filter postrouting
```

Figura 7.29 - Trace para ver el cambio inverso de NAT

```
chain prerouting {  
    type filter hook prerouting priority 0; policy accept;  
    ip saddr 8.8.8.8 ip daddr 192.168.106.137 counter packets 0 bytes 0  
}  
  
chain postrouting {  
    type filter hook postrouting priority 0; policy accept;  
}  
}  
table netdev filteringress {  
    chain ens33 {  
        type filter hook ingress device ens33 priority 0; policy accept;  
        ip protocol icmp counter packets 6 bytes 504 nftrace set 1  
        ip daddr 192.168.106.137 counter packets 2 bytes 168  
        ip saddr 8.8.8.8 ip daddr 192.168.106.137 counter packets 1 bytes 84  
    }  
}
```

Figura 7.30 - Contadores para comprobar cambio inverso de NAT

Con esto, quedaría visto la diferencia entre los dos tipos de NAT y el funcionamiento que estos tienen.

Llegados a este punto, se ha de explicar una aclaración acerca de la traducción de direcciones:

Para poder realizar un Stateful NAT, es imprescindible que se pueda realizar el seguimiento de conexión del paquete, debido a que la única opción para poder entrar en la tabla nat y en las cadenas de esta (cadenas tipo nat), es que se pueda realizar el seguimiento de la conexión de un paquete y que este pertenezca a un flujo o bien que pueda crearlo.

En el siguiente ejemplo, se podrá observar que al poner en la regla de ingress el argumento *notrack*, para que no se realice el seguimiento de la conexión, este paquete no pasará por las cadenas de nat, mientras que al quitar la regla, se podrá ver como vuelve a transitar por estas cadenas.

1. [Equipo:Encaminador - Usuario:root] Se borran las reglas de las dos cadenas de ingress y se sustituyen por las siguientes reglas.

```
Equipos:Encaminador Usuario:root

# Se borran las reglas de las cadenas de ingress

$ nft flush chain ens33
$ nft flush chain ens38

#Se añaden las reglas para activar la traza nftrace pero añadiendo
la opción notrack

$ nft add rule netdev filteringress ens33 ip protocol icmp meta
nftrace set 1 notrack
$ nft add rule netdev filteringress ens38 ip protocol icmp meta
nftrace set 1 notrack
```

2. [Equipo:PC1 - Usuario:dit] Se vuelve a realizar un ping desde PC1 a la dirección de la máquina pública con Ip 8.8.8.8

3. [Equipo:Encaminador - Usuario:root] Se revisa el camino del paquete mostrado por el monitor *nftrace*.

Como se puede ver en la Figura 7.31, existe una regla en el *hook ingress* con el argumento *notrack*, de forma que el paquete transita por el *hook ingress* y las cadenas *inet prerouting* e *inet input*, saltándose la cadena *ip nat prerouting*.

```
root@localhost:/home/dit# nft monitor trace
trace id 30cd2431 netdev ingr ens38 packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 0
th 84 icmp type echo-request icmp code 0 icmp id 6053 icmp sequence 1
trace id 30cd2431 netdev ingr ens38 rule ip protocol icmp nftrace set 1 notrack (verdict continue)
trace id 30cd2431 netdev ingr ens38 verdict continue
trace id 30cd2431 netdev ingr ens38 No está la cadena ip nat prerouting
trace id 30cd2431 inet filter prerouting verdict continue
trace id 30cd2431 inet filter prerouting
trace id 30cd2431 inet filter input packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 0
th 84 icmp type echo-request icmp code 0 icmp id 6053 icmp sequence 1
trace id 30cd2431 inet filter input rule ip daddr 10.10.1.10 counter packets 80 bytes 5977 (verdi
trace id 30cd2431 inet filter input verdict continue
trace id 30cd2431 inet filter input
trace id ef4fb4a0 inet filter output packet; oif "ens38" ip saddr 10.10.1.10 ip daddr 10.10.1.20 ip
quence 1
trace id ef4fb4a0 inet filter output rule ip protocol icmp nftrace set 1 (verdict continue)
trace id ef4fb4a0 inet filter output verdict continue
trace id ef4fb4a0 inet filter output
trace id ef4fb4a0 inet filter postrouting verdict continue
trace id ef4fb4a0 inet filter postrouting
^C
root@localhost:/home/dit#
```

Figura 7.31 - Argumento notrack y carencia de cadena nat

4. [Equipo:Encaminador - Usuario:root] Se borran las reglas de las dos cadenas de ingress y se vuelven a sustituir por las mismas, pero sin la opción *notrack*.

Equipos: Encaminador **Usuario:** root

```
# Se borran las reglas de las cadenas de ingress

$ nft flush chain ens33
$ nft flush chain ens38

#Se añaden las reglas para activar la traza nftrace

$ nft add rule netdev filteringress ens33 ip protocol icmp meta
nftrace set 1
$ nft add rule netdev filteringress ens38 ip protocol icmp meta
nftrace set 1
```

5. [Equipo:PC1 - Usuario:dit] Se vuelve a realizar un ping desde PC1 a la dirección de la máquina pública con Ip 8.8.8.8
6. [Equipo:Encaminador - Usuario:root] Se revisa el camino del paquete mostrado por el monitor *ntrace*.

```
root@localhost:/home/dit# nft monitor trace
trace id 08d66b44 netdev ingr ens38 packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29
th 84 icmp type echo-request icmp code 0 icmp id 6101 icmp sequence 1
trace id 08d66b44 [netdev inar ens38] rule ip protocol icmp nftrace set 1 (verdict continue)
trace id 08d66b44 netdev ingr ens38 verdict continue
trace id 08d66b44 netdev inar ens38
trace id 08d66b44 [inet filter prerouting] verdict continue
trace id 08d66b44 [inet filter prerouting]
trace id 08d66b44 ip nat prerouting verdict continue
trace id 08d66b44 ip nat prerouting [Se vuelve a pasar por las cadenas de tipo nat]
trace id 08d66b44 inet filter input packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29
th 84 icmp type echo-request icmp code 0 icmp id 6101 icmp sequence 1
trace id 08d66b44 inet filter input rule ip daddr 10.10.1.10 counter packets 81 bytes 6061 (verdict continue)
trace id 08d66b44 inet filter input verdict continue
trace id 08d66b44 inet filter input
trace id e5e6baf3 inet filter output packet: oif "ens38" ip saddr 10.10.1.10 ip daddr 10.10.1.20 ip ds
cquence 1
trace id e5e6baf3 inet filter output rule ip protocol icmp nftrace set 1 (verdict continue)
trace id e5e6baf3 inet filter output verdict continue
trace id e5e6baf3 inet filter output
trace id e5e6baf3 inet filter postrouting verdict continue
trace id e5e6baf3 inet filter postrouting
^C
root@localhost:/home/dit#
```

Figura 7.32 - Tras quitar argumento notrack, cadena prerouting tipo nat

Tras quitar este argumento, en la Figura 7.32, se puede volver a ver como el paquete vuelve a transitar por la cadena ip nat prerouting.

De esta forma, si se recibe algún paquete que no pertenezca a un flujo de datos o cuyo estado sea marcado como *untracked* o *invalid*, será imposible aplicarle el Stateful NAT.

Conclusión del ejemplo:

En este ejemplo, ha podido comprobarse la potencia del Stateful NAT.

Para poder usar este tipo de traducción de direcciones, es necesario que el módulo de conntrack, esté activo, ya que la traducción de direcciones se aplica por flujo de comunicación.

De forma que solo el primer paquete de un flujo de comunicación, hace *match* con la regla de la cadena nat, aplicándosele este cambio a todos los paquetes que pertenecen a ese flujo de comunicación.

También recordar, que si el paquete es de tipo inválido, no entrará en las cadenas nat, ni estará asociado a ningún flujo, por lo que no podría aplicársele este tipo de traducción de direcciones.

Para terminar con este capítulo, se realizarán 3 ejemplos de prueba;

- El primero de ellos, consiste en ver el uso de redirect en el *hook prerouting*
- El segundo de ellos, verá el uso de redirect en el *hook output*.
- Por último, se realizará una prueba de NAT en un escenario asimétrico, para comparar el uso del Stateful y el Stateless NAT.

7.4 Ejemplo Redirect en Prerouting

Descripción del ejemplo:

En este ejemplo se probará a realizar redirect en el *hook prerouting* y comprobar las respuestas recibidas y como se realiza la traducción de las direcciones en este ejemplo.

Para ello, se usará el escenario de 2 equipos “Anexo E - Escenario 2 Equipos”.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Se creará la siguiente estructura de cadenas nftables en Encaminador.
 - 1) Se crea una tabla filter de tipo inet
 - Esta tabla contendrá las cadenas; prerouting, input, forward, postrouting y output
 - 2) Se crea una tabla nat de tipo ip
 - Esta tabla contendrá las cadenas; prerouting y postrouting
 - 3) Se crea una tabla ingr de tipo netdev, la cual tendrá dos cadenas.
 - La cadena ingress, asociado a la interfaz “lo”
 - La cadena ens38, asociada a la interfaz del mismo nombre.

```
Equipo:Encaminador Usuario:root

#Creación de las tablas

$ nft add table inet filter
$ nft add table netdev ingr
$ nft add table ip nat
```

```

#Creación de las cadenas de la tabla nat

$ nft add chain ip nat prerouting '{ type nat hook prerouting priority -10 ; }'
$ nft add chain ip nat postrouting '{ type nat hook postrouting priority 10 ; }'

#Creación de las cadenas de la tabla filter

$ nft add chain inet filter postrouting '{ type filter hook postrouting priority 0 ; }'
$ nft add chain inet filter prerouting '{ type filter hook prerouting priority 0 ; }'
$ nft add chain inet filter forward '{ type filter hook forward priority 0 ; }'
$ nft add chain inet filter input '{ type filter hook input priority 0 ; }'
$ nft add chain inet filter output '{ type filter hook output priority 0 ; }'

#Creación de las cadenas de la tabla ingr

$ nft add chain netdev' ingr ingress { type filter hook ingress device lo priority 0 ; }
$ nft add chain netdev' ingr ens38 { type filter hook ingress device ens38 priority 0 ; }

```

2. **[Equipo:Encaminador - Usuario:root]** A continuación, se añadirán las siguientes reglas al *hook prerouting* de nat, de forma que se realice el Redirect según el protocolo y se contabilicen los paquetes del mismo.

Equipo:Encaminador **Usuario:**root

```

$ nft add rule ip nat prerouting ip protocol icmp counter redirect
$ nft add rule ip nat prerouting ip protocol tcp counter redirect
$ nft add rule ip nat prerouting ip protocol udp counter redirect

```

3. **[Equipo:Encaminador - Usuario:root]** A la tabla ingr, se le añaden las siguientes reglas de forma que se pueda realizar el seguimiento de los paquetes.

Equipo:Encaminador **Usuario:**root

```

$ nft add rule netdev ingr ingress nftrace set 1 counter
$ nft add rule netdev ingr ens38 ip protocol icmp nftrace set 1
counter

```

4. **[Equipo:Encaminador - Usuario:root]** Se activa el monitor de nftrace y el de conntrack.

Equipo:Encaminador **Usuario:**root

```

#Terminal 1
$ nft monitor trace

#Terminal 2

```

```
$ conntrack -E --event-mask ALL
```

5. [Equipo:PC1- Usuario:dit] Se comenzará probando el tráfico ICMP, para ello, PC1 realizará un ping a la dirección Ip 8.8.8.8.

```
Equipo:PC1 Usuario:dit
```

```
$ ping -c 1 8.8.8.8
```

6. [Equipo:PC1- Usuario:root] Se activa Tcpdump en PC1, para que capture tráfico en la interfaz “ens37”.

```
Equipo:PC1 Usuario:root
```

```
$ tcpdump -i ens37
```

7. [Equipo:Encaminador - Usuario:root] Se observa en el monitor de *ntrace* el camino seguido por el paquete.

```
root@localhost:/home/dit# nft monitor trace
trace id a5b024ea netdev ingr ens38 packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:le ip saddr 10.10.1.20 ip daddr 8.8.8.8 ip dscp cs0 ip e
84 icmp type echo-request icmp code 0 icmp id 3640 icmp sequence 1
trace id a5b024ea netdev ingr ens38 rule ip protocol icmp nftrace set 1 counter packets 9 bytes 756 (verdict continue)
trace id a5b024ea netdev ingr ens38 verdict continue
trace id a5b024ea netdev ingr ens38
trace id a5b024ea inet filter prerouting verdict continue
trace id a5b024ea ip nat prerouting
trace id a5b024ea ip nat prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:le ip saddr 10.10.1.20 ip daddr 8.8.8.8 ip dscp cs0 ip ecn n
cmp type echo-request icmp code 0 icmp id 3640 icmp sequence 1
trace id a5b024ea ip nat prerouting rule ip protocol icmp counter packets 7 bytes 588 redirect (verdict accept)
trace id a5b024ea inet filter input packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:le ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 i
th 84 icmp type echo-request icmp code 0 icmp id 3640 icmp sequence 1
trace id a5b024ea inet filter input rule ip daddr 10.10.1.10 counter packets 9 bytes 756 (verdict continue)
trace id a5b024ea inet filter input verdict continue
trace id af3af80 inet filter input
trace id af3af80 inet filter output packet: oif "ens38" ip saddr 10.10.1.10 ip daddr 10.10.1.20 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 51136 ip length 84 icmp type
quence 1
trace id af3af80 inet filter output rule ip protocol icmp nftrace set 1 (verdict continue)
trace id af3af80 inet filter output verdict continue
trace id af3af80 inet filter output
trace id af3af80 inet filter postrouting verdict continue
trace id af3af80 inet filter postrouting
`C
root@localhost:/home/dit#
```

Figura 7.33 - Camino ping PC1 redirect en prerouting

En la Figura 7.33, se puede ver el camino tanto del request como del reply ICMP.

- 1) El paquete ICMP sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena ens38 de la tabla ingr, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en la cadena ens38 de ingr es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 4) El paquete llega al *hook prerouting*, donde hace *match* con la regla de ICMP y se le hace redirect al paquete, siendo su veredicto en esta cadena de *continue*.
- 5) Se consulta el encaminamiento del paquete, determinando que este es para un proceso local (debido al uso de redirect).
- 6) El paquete continúa su camino por el *hook input*, siendo su veredicto *continue*.
- 7) El proceso local genera un nuevo paquete, el cual sale por output, siendo este la respuesta al ICMP-request enviado por PC1 hacia la máquina pública con Ip 8.8.8.8
- 8) El paquete sigue su curso por el *hook postrouting* y sale del equipo Encaminador por la interfaz “ens38”, hacia el equipo PC1.

8. [Equipo:PC1 - Usuario:root] Se observa la captura mostrada por Tcpdump en PC1.

```
[root@localhost dit]# tcpdump -i ens37
dropped privs to tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens37, link-type EN10MB (Ethernet), capture size 262144 bytes
02:26:05.239334 IP localhost.localdomain > 8.8.8.8: ICMP echo request, id 3899, seq 1, length 64
02:26:05.240128 IP gateway > localhost.localdomain: ICMP echo reply, id 3899, seq 1, length 64
```

Figura 7.34 - Captura PC1 ping redirect en prerouting

En la Figura 7.34, se puede observar que en primer lugar, se registra un paquete ICMP con origen local y destino a la máquina pública con Ip 8.8.8.8, teniendo este paquete ID 3899.

En la siguiente línea, se puede ver un paquete con Ip origen, nuestro Gateway (Encaminador), con Ip destino a PC1, siendo un paquete ICMP reply y teniendo el mismo ID, que el generado en la petición.

De forma que, se ha podido comprobar como con Redirect, la consulta ICMP es respondida por el equipo Encaminador, en lugar de la máquina pública con Ip 8.8.8.8

A pesar de que el origen del echo-reply, es distinto del que esperaría el ping de PC1 que sería 8.8.8.8 , este recibe el paquete y da el ping como válido.

9. [Equipo:Encaminador - Usuario:root] A continuación, se comprueba que se haya creado la entrada NAT en el sistema de conntrack.

Equipo:Encaminador Usuario:root

```
$ conntrack -L -j
```

Con este comando, le pedimos a conntrack que nos muestre las entradas que se hayan generado usando cualquier módulo NAT, tal y como se puede ver en la Figura 7.35, este nos muestra la entrada con el redirect aplicado.

Esto es así, puesto que vemos que en los valores del paquete request, se puede comprobar que la pareja de Ip's son: src:10.10.1.20 y dst:8.8.8.8.

Mientras que en los valores del paquete de respuesta, vemos que la pareja de Ip's son: src 10.10.1.10 y dst 10.10.1.20.

```
root@localhost:/home/dit# conntrack -L -j
icmp      1 26 src=10.10.1.20 dst=8.8.8.8 type=8 code=0 id=3658 src=10.10.1.10 dst=10.10.1.20 type=0 code=0 id=3658 mark=0 use=1
conntrack v1.4.6 (conntrack-tools): 1 flow entries have been shown.
root@localhost:/home/dit#
```

Figura 7.35 - Entrada conntrack NAT ICMP redirect en prerouting

También se puede observar este cambio si se realizan sucesivos envíos de ICMP en el monitor de conntrack, tal y como se puede observar en la Figura 7.36.

```
root@localhost:/home/dit# conntrack -E --any-nat
[DESTROY] icmp      1 src=10.10.1.20 dst=8.8.8.8 type=8 code=0 id=3640 src=10.10.1.10 dst=10.10.1.20 type=0 code=0 id=3640
[NEW]     icmp      1 30 src=10.10.1.20 dst=8.8.8.8 type=8 code=0 id=3658 [UNREPLIED] src=10.10.1.10 dst=10.10.1.20 type=0 code=0 id=3658
[UPDATE]  icmp      1 30 src=10.10.1.20 dst=8.8.8.8 type=8 code=0 id=3658 src=10.10.1.10 dst=10.10.1.20 type=0 code=0 id=3658
```

Figura 7.36 - Monitor conntrack NAT ICMP redirect en prerouting

Del mismo modo se comprueba ahora con el protocolo TCP, el redirect en la cadena prerouting y si al igual que ocurre con ICMP, se da por válido el cambio de destino.

10. [Equipo:PC1 - Usuario:root] Se lanza el comando curl desde PC1.

```
Equipo:PC1 Usuario:dit
$ curl 142.250.186.78
```

11. [Equipo:Encaminador - Usuario:root] Se observa en el monitor de *nftrace* el camino seguido por el paquete TCP.

```
add rule inet filter output ip protocol tcp nftrace set 1 counter
monitor trace
ens38 packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 142.250.186.78 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 4339 ip
dport http tcp flags == syn tcp window 29200
ens38 rule ip protocol tcp nftrace set 1 counter packets 7 bytes 420 (verdict continue)
ens38 verdict continue
ens38
prerouting verdict continue
prerouting
in packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 142.250.186.78 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 43394 ip
rt http tcp flags == syn tcp window 29200
in rule ip protocol tcp counter packets 2 bytes 120 redirect (verdict accept)
input packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 43394 ip
!l http [tcp flags == syn] ip window 29200
input rule ip daddr 10.10.1.10 counter packets 21 bytes 1596 (verdict continue)
input verdict continue
input
output packet: oif "ens38" ip saddr 10.10.1.10 ip daddr 10.10.1.20 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 0 ip length 60 tcp sport http tcp dport 57904 [tcp flags == 0x12] ip
output rule ip protocol tcp nftrace set 1 counter packets 0 bytes 0 (verdict continue)
output
postrouting verdict continue
postrouting
```

Figura 7.37 - Camino TCP redirect en prerouting

En la Figura 7.37, se puede ver el camino del paquete TCP con el flag SYN a Encaminador

- 1) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena ens38 de la tabla ingr, la cual activa la marca de seguimiento del paquete.
- 2) El veredicto en la cadena ens38 de ingr es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 3) El paquete llega al *hook prerouting*, donde hace *match* con la regla de TCP y se le hace redirect al paquete, siendo el veredicto en esta cadena de *continue*.
- 4) Se consulta el encaminamiento del paquete, determinando que este es para un proceso local (debido al uso de redirect).
- 5) El paquete continúa su camino por el *hook input*, siendo su veredicto *continue*.
- 6) El proceso local genera un nuevo paquete TCP, con los flags SYN-ACK, el cual sale por *output*.
- 7) El paquete sigue su curso por el *hook postrouting* y sale del equipo Encaminador por la interfaz “ens38”, hacia el equipo PC1.

12. [Equipo:Encaminador - Usuario:root] Pasamos a comprobar el seguimiento de conexiones de Encaminador.

Se puede observar que en la Figura 7.38, se crea una entrada en conntrack, con el estado de TCP SYN_SENT, además se puede ver la actualización de esta entrada con la respuesta SYN_RECV con origen en 10.10.1.10 y destino 10.10.1.20.

Pero más allá de esto, nunca se recibe el TCP ACK de PC1, por lo que tras transcurrir el tiempo de vida de 60 segundos desde la última actualización de la entrada, esta es destruida.

```

root@localhost:/home/dit# conntrack -E -j
[NEW] tcp      6 120 SYN_SENT src=10.10.1.20 dst=142.250.186.78 sport=57902 dport=80 [UNREPLIED] src=10.10.1.10 dst=10.10.1.20 sport=80 dport=57902
[UPDATE] tcp    6 60 SYN_RECV src=10.10.1.20 dst=142.250.186.78 sport=57902 dport=80 src=10.10.1.10 dst=10.10.1.20 sport=80 dport=57902
[UPDATE] tcp    6 60 SYN_RECV src=10.10.1.20 dst=142.250.186.78 sport=57902 dport=80 src=10.10.1.10 dst=10.10.1.20 sport=80 dport=57902
[UPDATE] tcp    6 60 SYN_RECV src=10.10.1.20 dst=142.250.186.78 sport=57902 dport=80 src=10.10.1.10 dst=10.10.1.20 sport=80 dport=57902
[DESTROY] tcp   6 src=10.10.1.20 dst=142.250.186.78 sport=57902 dport=80 src=10.10.1.10 dst=10.10.1.20 sport=80 dport=57902

```

Figura 7.38 - Monitor conntrack TCP redirect en prerouting

Por lo que, a diferencia de ICMP, el hecho de que la respuesta venga del equipo Encaminador, no sirve para validar la petición de curl.

Por último, nos queda el protocolo UDP.

13. [Equipo:PC1 - Usuario:dit] Para probar el protocolo UDP desde PC1, se usa el siguiente comando.

```

Equipo:PC1 Usuario:dit

$ nslookup 8.8.8.8

```

14. [Equipo:Encaminador - Usuario:root] Se observa en el monitor de *ntrace* el camino seguido por el paquete UDP.

```

trace id a83c01fe netdev ingr ens38 packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:le ip saddr 10.10.1.20 ip daddr 8.8.8.8 ip dscp cs0 ip
66 udp sport 53993 udp dport domain udp length 46
trace id a83c01fe netdev ingr ens38 rule ip protocol udp nftrace set 1 counter packets 2 bytes 149 (verdict continue)
trace id a83c01fe netdev ingr ens38 Verdicht continue
trace id a83c01fe netdev ingr ens38
trace id a83c01fe inet filter prerouting verdict continue
trace id a83c01fe inet filter prerouting
trace id a83c01fe ip nat prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:le ip saddr 10.10.1.20 ip daddr 8.8.8.8 ip dscp cs0 ip ecn
dp sport 53993 udp dport domain udp length 46
trace id a83c01fe ip nat prerouting rule ip protocol udp counter packets 2 bytes 135 redirect (verdict accept)
trace id a83c01fe inet filter input packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:le ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0
th 66 udp sport 53993 udp dport domain udp length 46
trace id a83c01fe inet filter input rule ip daddr 10.10.1.10 counter packets 26 bytes 1912 (verdict continue)
trace id a83c01fe inet filter input verdict continue
trace id a83c01fe inet filter input
trace id e3c7a184 inet filter output packet: oif "ens38" ip saddr 10.10.1.10 ip daddr 10.10.1.20 ip dscp cs6 ip ecn not-ect ip ttl 64 ip id 20039 ip length 94 icmp typ
0 icmp sequence 0
trace id e3c7a184 inet filter output rule ip protocol icmp nftrace set 1 (verdict continue)
trace id e3c7a184 inet filter output verdict continue
trace id e3c7a184 inet filter output
trace id e3c7a184 inet filter postrouting verdict continue
trace id e3c7a184 inet filter postrouting

```

Figura 7.39 - Camino UDP redirect en prerouting

En la Figura 7.39, se puede ver el camino del paquete UDP.

- 1) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena ens38 de la tabla ingr, la cual activa la marca de seguimiento del paquete.
- 2) El veredicto en la cadena ens38 de ingr es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 3) El paquete llega al *hook prerouting*, donde hace *match* con la regla de UDP y se le hace redirect al paquete, siendo el veredicto en esta cadena de *continue*.
- 4) Se consulta el encaminamiento del paquete, determinando que este es para un proceso local (debido al uso de redirect).
- 5) El paquete continúa su camino por el *hook input*, siendo su veredicto *continue*.
- 6) Al no existir un proceso local (Servidor DNS), que pueda responder a esta petición, se genera un paquete ICMP para avisar al equipo PC1
- 7) El paquete sigue su curso por el *hook postrouting* y sale del equipo Encaminador por la interfaz “ens38”, hacia el equipo PC1.

15. [Equipo:Encaminador - Usuario:root] Pasamos a comprobar el seguimiento de conexiones de Encaminador.

```
[NEW] udp      17 30 src=10.10.1.20 dst=8.8.8.8 sport=52103 dport=53 [UNREPLIED] src=10.10.1.10 dst=10.10.1.20 sport=53 dport=52103
[NEW] udp      17 30 src=10.10.1.20 dst=8.8.8.8 sport=53993 dport=53 [UNREPLIED] src=10.10.1.10 dst=10.10.1.20 sport=53 dport=53993
[NEW] udp      17 30 src=10.10.1.20 dst=8.8.8.8 sport=38435 dport=53 [UNREPLIED] src=10.10.1.10 dst=10.10.1.20 sport=53 dport=38435
[NEW] udp      17 30 src=10.10.1.20 dst=8.8.8.8 sport=40838 dport=53 [UNREPLIED] src=10.10.1.10 dst=10.10.1.20 sport=53 dport=40838
[DESTROY] udp     17 src=10.10.1.20 dst=8.8.8.8 sport=54009 dport=53 [UNREPLIED] src=10.10.1.10 dst=10.10.1.20 sport=53 dport=54009
[DESTROY] udp     17 src=10.10.1.20 dst=8.8.8.8 sport=53993 dport=53 [UNREPLIED] src=10.10.1.10 dst=10.10.1.20 sport=53 dport=53993
[DESTROY] udp     17 src=10.10.1.20 dst=8.8.8.8 sport=52103 dport=53 [UNREPLIED] src=10.10.1.10 dst=10.10.1.20 sport=53 dport=52103
[DESTROY] udp     17 src=10.10.1.20 dst=8.8.8.8 sport=38435 dport=53 [UNREPLIED] src=10.10.1.10 dst=10.10.1.20 sport=53 dport=38435
[DESTROY] udp     17 src=10.10.1.20 dst=8.8.8.8 sport=40838 dport=53 [UNREPLIED] src=10.10.1.10 dst=10.10.1.20 sport=53 dport=40838
```

Figura 7.40 - Monitor conntrack UDP redirect prerouting

Como se puede ver en las Figura 7.40, la petición llega al equipo tras aplicarse la regla redirect.

Encaminador, al no tener instalado ningún servidor DNS local, responde con un paquete ICMP de respuesta, de forma que la entrada que se crea de UDP nunca es actualizada y es destruida tras pasar su tiempo de vida.

Si se instala un servidor DNS local en Encaminador y se vuelve a realizar la prueba, se puede ver como este contesta y la entrada de conntrack se actualiza y borra su flag de UNREPLIED, funcionando la consulta de PC1, tal y como se puede ver en la Figura 7.41.

```
[NEW] udp      17 30 src=10.10.1.20 dst=8.8.8.8 sport=54009 dport=53 [UNREPLIED] src=10.10.1.10 dst=10.10.1.20 sport=53 dport=54009
[UPDATE] udp     17 30 src=10.10.1.20 dst=8.8.8.8 sport=53993 dport=53 src=10.10.1.10 dst=10.10.1.20 sport=53 dport=54009
```

Figura 7.41 - Actualizada entrada UDP redirect en prerouting

Por lo que, el único protocolo para el cual no ha funcionado redirect, es TCP, puesto que para UDP e ICMP, se ha realizado el cambio de destino y PC1 ha aceptado las respuestas enviadas por Encaminador.

16. [Equipo:Encaminador - Usuario:root] Por último, se muestra la estructura de nftables en la Figura 7.42, la cuál ha sido usada para realizar las pruebas de este ejemplo

Equipo:Encaminador Usuario:root

```
$ nft list ruleset
```

```

table ip nat {
    chain prerout {
        type nat hook prerouting priority 0; policy accept;
        ip protocol icmp counter packets 12 bytes 1008 redirect
        ip protocol tcp counter packets 3 bytes 180 redirect
        ip protocol udp counter packets 2 bytes 135 redirect
    }

    chain out {
        type nat hook output priority 10; policy accept;
        ip protocol icmp counter packets 7 bytes 588 redirect
    }
}
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
        ip daddr 127.0.0.1 counter packets 0 bytes 0
        ip daddr 10.10.1.10 counter packets 26 bytes 1912
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
        ip daddr 127.0.0.1 counter packets 0 bytes 0 continue
        counter packets 49 bytes 3584
    }

    chain output {
        type filter hook output priority 0; policy accept;
        ip protocol icmp nftrace set 1
        ip protocol tcp nftrace set 1 counter packets 9 bytes 540
    }

    chain prerouting {
        type filter hook prerouting priority 0; policy accept;
    }

    chain postrouting {
        type filter hook postrouting priority 0; policy accept;
    }
}

```

Figura 7.42 - Estructura nftables Redirect en prerouting

Conclusión del ejemplo:

Como se ha podido ver en este ejemplo, el uso de *redirect* en el *hook prerouting*, resulta muy útil para ocultar servicios, o para redirigir ciertas comunicaciones hacia el equipo local, en lugar de encaminarlos hacia otro punto.

Debemos tener cuidado con este ejemplo, ya que como se ha podido comprobar, no todos los protocolos aceptan respuestas de este tipo de redirecciones, de forma que, si la respuesta no viene del equipo al que se le realiza la pregunta, no se considera válido y por lo tanto no se lleva a cabo la comunicación.

7.5 Ejemplo Redirect en Output

Descripción del ejemplo:

En este ejemplo se usará la acción *redirect*, pero en esta ocasión en el *hook output* y se comprobará las respuestas recibidas y como se realiza la traducción de las direcciones en este ejemplo.

Se usará el escenario de 2 equipos “Anexo E - Escenario 2 Equipos”.

Como se ha podido comprobar, si *redirect* está en *output*, no afecta a los mensajes de respuesta que envíe el kernel de Encaminador, ya que estos o bien pertenecen a un flujo de conexiones o no atraviesan las cadenas pertenecientes a NAT.

Aun así, se volverá a dejar constancia de esto con un ejemplo sencillo.

Análisis del ejemplo:

1. [Equipo:Encaminador - Usuario:root] Se crea la siguiente estructura en nftables.

- 1) Se crea una tabla filter de tipo inet
 - Esta tabla contendrá las cadenas; input, forward y output
- 2) Se crea una tabla nat de tipo ip
 - Esta tabla contendrá la cadena output
- 3) Se crea una tabla ingr de tipo netdev.
 - Esta tabla tendrá la cadena ens38, asociada a la interfaz del mismo nombre.

```
Equipo:Encaminador Usuario:root
```

```
#Creación de tablas
$ nft add table filter
$ nft add table netdev ingr
$ nft add table ip nat

#creación de cadenas de filter

$ nft add chain inet filter forward '{ type filter hook forward priority 0 ; }'
$ nft add chain inet filter input '{ type filter hook input priority 0 ; }'
$ nft add chain inet filter output '{ type filter hook output priority 0 ; }'

#creación de cadenas de nat
$ nft add chain ip nat output '{ type nat hook output priority 2 ; }'

#creación de la cadena ens38
$ nft add chain netdev' ingr ens38 { type filter hook ingress device ens38 priority 0 ; }
```

2. [Equipo:Encaminador - Usuario:root] A continuación, se añadirán las siguientes reglas:

- En la cadena output de nat, se añade la regla que realice el redirect del protocolo ICMP.
- En la cadena output de filter, se añaden dos reglas, una que active la marca *nftrace* de los paquetes y otra que lleve la cuenta de los paquetes que atraviesan por ella.
- Por último, se añade una regla a la cadena ens38, que active la marca *nftrace* del tráfico ICMP.

```
Equipo:Encaminador Usuario:root
```

```
$ nft add rule ip nat output ip protocol icmp counter continue
$ nft add rule ip nat output ip protocol icmp counter redirect
$ nft add rule inet filter output nftrace set 1
$ nft add rule inet filter output ip protocol icmp counter continue
$ nft add rule netdev ingr ens38 ip protocol icmp nftrace set 1
counter
```

3. [Equipo:Encaminador - Usuario:root] Se comprueba el *ruleset* de nftables para ver que las reglas y la estructura se han creado correctamente.

Además, se activa el monitor de *nftrace*.

La salida del comando *list*, puede comprobarse en la Figura 7.43.

```
Equipo:Encaminador Usuario:root
```

```
#Terminal 1
$ nft list ruleset

#Terminal 2
$ nft monitor trace
```

```
root@localhost:/home/dit# nft list ruleset
table ip nat {
    chain output {
        type nat hook output priority 2; policy accept;
        ip protocol icmp counter packets 1 bytes 84 continue
        ip protocol icmp counter packets 0 bytes 0 redirect
    }
}
table inet filter {
    chain input {
        type filter hook input priority 0; policy accept;
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
        nftrace set 1 continue
        ip protocol icmp counter packets 3 bytes 252 continue
    }
}
table netdev ingr {
    chain ens38 {
        type filter hook ingress device ens38 priority 0; policy accept;
        ip protocol icmp nftrace set 1 counter packets 1 bytes 84
    }
}
```

Figura 7.43 - Estructura redirect en output

4. [Equipo:PC1 - Usuario:dit] Se lanza un ping desde PC1 hacia Encaminador.

```
Equipo:PC1 Usuario:dit
```

```
$ ping -c 1 10.10.1.10
```

5. [Equipo:Encaminador - Usuario:root] Se observa en el monitor de *nftrace* el camino seguido por el paquete ICMP.

```

root@localhost:/home/dit# nft monitor trace
trace id aa9546fc netdev ingr ens38 packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad
ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip e
cn not-ect ip ttl 64 ip id 23787 ip length 84 icmp type echo-request icmp code 0 icmp
id 3116 icmp sequence 1
trace id aa9546fc netdev ingr ens38 rule ip protocol icmp nftrace set 1 counter packet
s 0 bytes 0 (verdict continue)
trace id aa9546fc netdev ingr ens38 verdict continue
trace id aa9546fc netdev ingr ens38
trace id aa9546fc inet filter input verdict continue
trace id aa9546fc inet filter input
trace id a20d129c inet filter output packet: oif "ens38" ip saddr 10.10.1.10 ip daddr
10.10.1.20 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 55705 ip length 84 icmp type ech
o-reply icmp code 0 icmp id 3116 icmp sequence 1
trace id a20d129c inet filter output rule nftrace set 1 continue (verdict continue)
trace id a20d129c inet filter output rule ip protocol icmp counter packets 0 bytes 0 c
ontinue (verdict continue)
trace id a20d129c inet filter output verdict continue
trace id a20d129c inet filter output
^C
root@localhost:/home/dit#

```

Figura 7.44 - Camino ICMP externo redirect en output

En la Figura 7.44, se puede ver el camino del paquete ICMP enviado por PC1.

- 1) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena `ens38` de la tabla `ingr`, la cual activa la marca de seguimiento del paquete.
 - 2) El veredicto en la cadena `ens38` de `ingr` es *continue*, por lo que el paquete continúa su camino hacia la cadena `prerouting` de `filter`.
 - 3) El paquete llega al *hook input*, donde se toma como veredicto *continue*.
 - 4) El paquete llega al proceso local.
 - 5) El proceso local genera la respuesta, la cual llega al *hook output* y se le aplican las reglas de la cadena `output` de la tabla `filter`.
En este punto debemos observar que este paquete no ha entrado en la cadena `output` de la tabla `nat`, y esto se debe a que al ser un paquete de respuesta, este no atraviesa las cadenas de `nat`.
 6. **[Equipo:Encaminador - Usuario:root]** Se lanza un ping desde Encaminador hacia la máquina pública 8.8.8.8.
- Equipo:PC1 Usuario:dit**

```
$ ping -c 1 10.10.1.10
```
7. **[Equipo:Encaminador - Usuario:root]** Se observa en el monitor de `nftrace` el camino seguido por el paquete ICMP generado por Encaminador.

```

root@localhost:/home/dit# nft monitor trace
trace id 45e9e783 inet filter output packet: oif "ens33" ip saddr 192.168.106.137 ip daddr 8.8.8.8 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 59347 ip length 84 icmp type echo-request icmp code 0 icmp id 2509 icmp sequence 1
trace id 45e9e783 inet filter output rule nftrace set 1 continue (verdict continue)
trace id 45e9e783 inet filter output rule ip protocol icmp counter packets 3 bytes 252 continue (verdict continue)
trace id 45e9e783 inet filter output verdict continue
trace id 45e9e783 inet filter output
trace id 45e9e783 ip nat output packet: oif "ens33" ip saddr 192.168.106.137 ip daddr 8.8.8.8 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 59347 ip length 84 icmp type echo-request icmp code 0 icmp id 2509 icmp sequence 1
trace id 45e9e783 ip nat output rule ip protocol icmp counter packets 1 bytes 84 continue (verdict continue)
trace id 45e9e783 ip nat output rule ip protocol icmp counter packets 0 bytes 0 redirect (verdict accept)

```

Figura 7.45 - Camino ICMP interno redirect en output

En la Figura 7.45, se puede ver el camino del paquete ICMP enviado por Encaminador.

- 1) El paquete es generado por el proceso local, por lo que, tras tomar su decisión de encaminamiento, se dirige al *hook output*.
- 2) En el *hook output* de la tabla filter, hace *match* con las dos reglas, activando la marca *nftrace* del paquete y aumentando el contador de paquetes de la segunda regla, tras esto, se toma como veredicto *continue*.
- 3) El paquete continua por la cadena output de la tabla nat, donde hace *match* con el redirect y se toma como veredicto, el *accept* del paquete

Hacer redirect en output, implica que el camino que sigue el paquete, es en verdad el mismo camino que sigue los paquetes de la interfaz “lo” Figura 4.21.

Para este ejemplo, es más interesante asociar el *hook ingress* a la interfaz local.

8. **[Equipo:Encaminador - Usuario:root]** Se crea una nueva cadena ingress, asociada a la interfaz “lo”, y se añaden las cadenas prerouting y postrouting a la tabla filter.

Equipo:Encaminador Usuario:root

```

$ nft add chain netdev' ingr ingress { type filter hook ingress device lo priority 0 ; }
$ nft add rule netdev ingr ingress ip protocol icmp nftrace set 1 counter

#Se añaden las cadenas a la tabla filter

$ nft add chain inet filter postrouting '{ type filter hook postrouting priority 0 ; }'
$ nft add chain inet filter prerouting '{ type filter hook prerouting priority 0 ; }'

```

9. **[Equipo:Encaminador - Usuario:root]** Se activa Wireshark para que capture tráfico en la interfaz “lo”, se activa el monitor de conntrack y se realiza un ping a la dirección 8.8.8.8.

10. **[Equipo:Encaminador - Usuario:root]** Se observa en el monitor de *nftrace* el camino seguido por el paquete ICMP generado por Encaminador.

En este caso, al tener la nueva cadena de netdev asociada a la interfaz “lo”, se podrá visualizar mucho mejor el camino del paquete

```

trace id 329a4c5f ip nat out packet: oif "ens33" ip saddr 192.168.106.137 ip daddr 8.8.8.8 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 18427
ce 1
trace id 329a4c5f ip nat out rule ip protocol icmp counter packets 0 bytes 0 redirect (verdict accept)
trace id 329a4c5f inet filter postrouting verdict continue
trace id 329a4c5f inet filter postrouting
trace id 534bc2e6 netdev ingr ingress packet: iif "lo"
trace id 534bc2e6 netdev ingr ingress rule nftrace set 1 counter packets 20 bytes 1000 (verdict continue)
trace id 534bc2e6 netdev ingr ingress verdict continue
trace id 534bc2e6 netdev ingr ingress
trace id 534bc2e6 inet filter prerouting verdict continue
trace id 534bc2e6 inet filter prerouting
trace id 534bc2e6 inet filter input verdict continue
trace id 534bc2e6 inet filter input
trace id 47ab76df inet filter output packet: oif "lo" ip saddr 127.0.0.1 ip daddr 192.168.106.137 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id
equence 1
trace id 47ab76df inet filter output rule ip protocol icmp nftrace set 1 (verdict continue)
trace id 47ab76df inet filter output verdict continue
trace id 47ab76df inet filter output
trace id 47ab76df inet filter postrouting verdict continue
trace id 47ab76df inet filter postrouting
trace id 70d6722f netdev ingr ingress packet: iif "lo"
trace id 70d6722f netdev ingr ingress rule nftrace set 1 counter packets 20 bytes 1000 (verdict continue)
trace id 70d6722f netdev ingr ingress verdict continue
trace id 70d6722f netdev ingr ingress
trace id 70d6722f inet filter prerouting verdict continue
trace id 70d6722f inet filter prerouting
trace id 70d6722f inet filter input verdict continue
trace id 70d6722f inet filter input

```

Figura 7.46 - Camino completo ICMP redirect en output

En la Figura 7.46, se puede ver el camino del paquete ICMP enviado por Encaminador.

- 1) El paquete es generado por el proceso local, por lo que, tras tomar su decisión de encaminamiento, se dirige al *hook output*.
- 2) En el *hook output* de la tabla filter, hace *match* con las dos reglas, activando la marca *nftrace* del paquete y aumentando el contador de paquetes de la segunda regla, tras esto, se toma como veredicto *continue*.
- 3) El paquete continúa por la cadena output de la tabla nat, donde hace *match* con el redirect y se toma como veredicto, el *accept* del paquete
- 4) El paquete llega a la interfaz “lo”, tras esto, continúa su camino tal y como se vio en el ejemplo del ping local del capítulo 4.
- 5) El paquete llega al *hook ingress* asociado a la interfaz “lo”.
- 6) El paquete sigue su curso por las cadenas prerouting e input, hasta llegar al proceso local.
- 7) Este recibe el paquete request y genera el paquete ICMP reply, el cual, tras ser generado, toma su decisión de encaminamiento y se dirige hacia la interfaz “lo”.
- 8) El paquete al ser de respuesta y pertenecer a un flujo de comunicación ya creado, no pasa por la cadena ip nat, cruzando solo por las cadenas de la tabla filter, hasta llegar a la interfaz “lo”
- 9) Esta vuelve a conducir el paquete hacia la cadena ingress asociada a esta interfaz.
- 10) Tras esto, el paquete ICMP reply, cruza las cadenas de prerouting e input y finalmente llega al proceso local.

Como se puede comprobar en la Figura 7.46, el paquete comienza generándose en el kernel y tiene su salida por la interfaz “ens33”, tras aplicar la regla redirect en ip nat output, se puede ver que esta sigue su transcurso por postrouting y seguidamente entra la por la interfaz “lo” y a partir de este punto, sigue el mismo camino que el ya visto en la Figura 4.21.

Vemos que la respuesta generada, tiene dirección origen 127.0.0.1 y dirección destino 192.168.106.137 y que su interfaz de salida es “lo”, por la cual, transita para volver a entrar por prerouting y finalmente cruzar por input, hasta llegar la respuesta al kernel de Encaminador.

11. [Equipo:Encaminador - Usuario:root] Se observa la salida de Wireshark para comprobar que por cada ID ICMP se registran solo 2 entradas, una correspondiente al request y otra al reply.

En la Figura 7.47, se puede comprobar que el echo-request, sale por la interfaz “lo”, con dirección origen 192.168.106.137 y dirección destino 127.0.0.1, mientras que la respuesta pasa también por la interfaz “lo”, con Ip origen 127.0.0.1 e Ip destino 192.168.106.137.

Apply a display filter ... <Ctrl-/>						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.106.137	127.0.0.1	ICMP	98	Echo (ping) request id=0x082a, seq=1/256, ttl=64 (reply in 2)
2	0.000152634	127.0.0.1	192.168.106.137	ICMP	98	Echo (ping) reply id=0x082a, seq=1/256, ttl=64 (request in 1)
3	512.917694432	192.168.106.137	127.0.0.1	ICMP	98	Echo (ping) request id=0x085d, seq=1/256, ttl=64 (reply in 4)
4	512.917719648	127.0.0.1	192.168.106.137	ICMP	98	Echo (ping) reply id=0x085d, seq=1/256, ttl=64 (request in 3)

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
 ▶ Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)
 ▶ Internet Protocol Version 4, Src: 192.168.106.137, Dst: 127.0.0.1
 0100 = Version: 4
 0101 = Header Length: 20 bytes (5)
 ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 Total Length: 84
 Identification: 0xad42 (44354)
 Flags: 0x4000, Don't fragment
 Time to live: 64
 Protocol: ICMP (1)
 Header checksum: 0xe333 [validation disabled]
 [Header checksum status: Unverified]
 Source: 192.168.106.137
 Destination: 127.0.0.1
 ▶ Internet Control Message Protocol

Figura 7.47 - Captura tráfico interfaz “lo”, redirect en output

12. [Equipo:Encaminador - Usuario:root] Se comprueba la salida del monitor de conntrack.

Además de esto, se comprueba también la monitorización de conntrack (ver Figura 7.48), en la cual se puede ver, como se ha creado una entrada y que los campos pertenecientes a la respuesta que se espera ver, tienen aplicado el redirect (origen 127.0.0.1 y destino 192.168.106.137).

root@localhost:/home/dit# conntrack -E --event-mask ALL
[NEW] icmp 1 30 src=192.168.106.137 dst=8.8.8.8 type=8 code=0 id=2217 [UNREPLIED] src=127.0.0.1 dst=192.168.106.137 type=0 code=0 id=2217
[UPDATE] icmp 1 30 src=192.168.106.137 dst=8.8.8.8 type=8 code=0 id=2217 src=127.0.0.1 dst=192.168.106.137 type=0 code=0 id=2217
[DESTROY] icmp 1 src=192.168.106.137 dst=8.8.8.8 type=8 code=0 id=2217 src=127.0.0.1 dst=192.168.106.137 type=0 code=0 id=2217

Figura 7.48 - Entradas NAT conntrack , redirect en output

Conclusión del ejemplo:

En este ejemplo, se ha podido comprobar el uso de *redirect* en el *hook output*.

Como se vio al comienzo de este capítulo, en las cadenas de tipo nat, solo entran aquellos paquetes con estado válido que puedan abrir un flujo de comunicación.

De forma que, si por el *hook output* llega una respuesta, a esta no se le aplicará la traducción de direcciones, ya que, o bien es de tipo inválido y por lo tanto no entra en la cadena nat, o ya pertenece a un flujo de comunicación y tampoco se le aplicarían los cambios de esta cadena.

En el ejemplo, se puede ver que el camino que realizaría el paquete al que se le realiza el *redirect*, sigue el flow de paquetes de la interfaz “lo”, vista en el capítulo 4 “Escenario prueba flow interfaz lo”.

7.6 Ejemplo NAT Asimétrico

Descripción del ejemplo:

Este ejemplo busca ejemplificar como se ha de realizar la traducción de direcciones, en las cuales existen dos caminos, uno de ida para las peticiones y otro distinto de vuelta para las respuestas.

Para este ejemplo se usará el escenario de 3 equipos con el caso especial del uso del equipo Router Nat, para

poder tener un escenario de 4 equipos tal y como aparece en la Figura F.0.11 .

En este escenario PC1 lanzará una petición ICMP echo-request con dirección a PC2, esta petición viajará hasta el equipo Encaminador, el cual le realizará un Snat antes de salir a la subred 172.168.0.0/24, y este Snat lo hará a la dirección de la interfaz que tiene en esta subred el equipo Router NAT.

De forma que PC2 cursará el echo-reply por el equipo Router NAT, y aquí es donde se verá la dificultad de este escenario, y es que debido a que lo que llega es un mensaje de respuesta, este es incapaz de crear una entrada en conntrack, y tampoco existe entrada alguna en el equipo que permita que esta petición acceda a las cadenas nat para realizarle un Dnat y poder poner la ip de PC1 como la dirección de destino.

Para solventar esa dificultad, se decide realizar un Stateful NAT en el equipo Encaminador, ya que a él le llegan las peticiones, pues es el camino directo desde PC1, y un Stateless NAT en el equipo Router NAT.

Análisis del ejemplo:

1. [Equipo:Encaminador - Usuario:root] Se crean las siguientes tablas en nftables.

- Se crea una tabla nat
- Se crea una tabla filter
- Se crea una tabla filteringress

```
Equipo:Encaminador Usuario:root
```

```
$ nft add table ip nat
$ nft add table ip filter
$ nft add table netdev ingress
```

2. [Equipo:Encaminador - Usuario:root] Se crean las siguientes cadenas en nftables.

En la tabla nat se añaden las siguientes cadenas:

- Una cadena prerouting
- Una cadena postrouting

En la tabla filter se añaden las siguientes cadenas:

- Una cadena prerouting
- Una cadena postrouting
- Una cadena input
- Una cadena output
- Una cadena forward

En la tabla filteringress, solo se añade la cadena ingress, la cual está asociada a la interfaz ens38.

```
Equipo:Encaminador Usuario:root
```

```
#creación de cadenas de filter

$ nft add chain ip filter prerouting '{ type filter hook prerouting priority 0 ; }'
$ nft add chain ip filter postrouting '{ type filter hook postrouting priority 0 ; }'
```

```

$ nft add chain inet filter forward '{ type filter hook forward priority 0 ; }'
$ nft add chain inet filter input '{ type filter hook input priority 0 ; }'
$ nft add chain inet filter output '{ type filter hook output priority 0 ; }'

#creación de cadenas de nat
$ nft add chain ip nat prerouting '{ type nat hook prerouting priority -1 ; }'
$ nft add chain ip nat postrouting '{ type nat hook postrouting priority -1 ; }'

#creación de la cadena ingress
$ nft add chain netdev' filteringress ingress { type filter hook ingress device ens38 priority 0 ; }'

```

3. [Equipo:Encaminador - Usuario:root] Se crean las siguientes reglas en nftables.

- En la cadena postrouting de la tabla nat se añade una regla para que todos los paquetes, cuyo origen sea la Ip de PC1 , se traten con un Snat que cambie su dirección origen por la Ip 172.168.0.40 (Esta Ip es la que tiene la interfaz del equipo Router NAT).
- En la cadena filter se añaden las siguientes reglas, cada una con su correspondiente contador de paquetes.
 - Una regla que haga match con el tráfico ICMP cuyo state sea related
 - Una regla que haga match con el tráfico TCP cuyo state sea related
 - Una regla que haga match con el tráfico ICMP cuyo state sea invalid
 - Una regla que haga match con el tráfico TCP cuyo state sea invalid
 - Una regla que haga match con el tráfico TCP cuyo estado sea new o established
- En la cadena ingress, se añade una regla que active la marca nftrace.

Equipo:Encaminador Usuario:root

```

#reglas cadena postrouting nat
$ nft add rule ip nat postrouting ip saddr 10.10.1.20 counter snat to 172.168.0.40

#reglas cadena prerouting tabla filter
$ nft add rule ip filter prerouting ip protocol icmp ct state related counter
$ nft add rule ip filter prerouting ip protocol tcp ct state related counter
$ nft add rule ip filter prerouting ip protocol icmp ct state invalid counter
$ nft add rule ip filter prerouting ip protocol tcp ct state invalid counter
$ nft add rule ip filter prerouting ip protocol tcp ct state { established, new } counter

#reglas cadena ingress tabla filteringress

```

```
$ nft add rule inet filter output ip protocol icmp counter continue
$ nft add rule netdev filteringress ingress nftrace set 1 counter
```

4. [Equipo:Encaminador - Usuario:root] Se observa el ruleset de nftables.

Se puede comprobar que la salida de este comando coincide con la Figura 7.49

Equipo:Encaminador Usuario:root

```
$ nft monitor trace
```

```
table ip nat {
    chain postrouting {
        type nat hook postrouting priority -1; policy accept;
        ip saddr 10.10.1.20 counter packets 53 bytes 4235 snat to 172.168.0.40
    }
    chain prerouting {
        type nat hook prerouting priority -1; policy accept;
    }
}
table ip filter {
    chain prerouting {
        type filter hook prerouting priority -1; policy accept;
        ip protocol icmp ct state related counter packets 4 bytes 448 continue
        ip protocol tcp ct state related counter packets 0 bytes 0 continue
        ip protocol icmp ct state invalid counter packets 1 bytes 28 continue
        ip protocol tcp ct state invalid counter packets 5 bytes 200 continue
        ip protocol tcp ct state { established, new } counter packets 12243 bytes 108031528 continue
    }
    chain postrouting {
        type filter hook postrouting priority -1; policy accept;
    }
    chain output {
        type filter hook output priority -1; policy accept;
        nftrace set 1
        ip protocol icmp ct state invalid counter packets 0 bytes 0 continue
        ip protocol icmp ct state related counter packets 6 bytes 504 continue
    }
    chain input {
        type filter hook input priority -1; policy accept;
    }
    chain forward {
        type filter hook forward priority 0; policy accept;
    }
}
```

Figura 7.49 - Estructura nftables Encaminador, escenario asimétrico

Las comprobaciones que se pueden ver en la cadena prerouting se usan para poder seguir la trazabilidad los diferentes paquetes ICMP que se puedan enviar.

Además, se puede comprobar que en la cadena postrouting de nat se realiza un Snat con la dirección de la interfaz del equipo Router NAT.

5. [Equipo:Router NAT - Usuario:root] En el equipo Router NAT, se crea la siguiente estructura de nftables.

- 1) Se crea una tabla filter
- 2) Se crea una cadena prerouting dentro de la tabla filter
- 3) Se añade a la cadena, la regla para realizar un Stateless NAT, de forma que todos los paquetes con destino la 172.168.0.40 cambiarán su dirección destino por la Ip 10.10.1.20

Equipo:Router NAT Usuario:root

```
$ nft add table filter
$ nft add chain ip filter prerouting '{ type filter hook prerouting
priority -1 ; }'
$ nft add rule ip filter prerouting ip daddr 172.168.0.40 ip daddr
set 10.10.1.20 counter
```

6. [Equipo:Router NAT - Usuario:root] Mostramos el *ruleset* de nftables.

Podemos comprobar que su salida coincide con la mostrada en la Figura 7.50.

```
Equipo:Router NAT Usuario:root
```

```
$ nft list ruleset
```

```
[root@localhost dit]# nft list ruleset
table ip filter {
    chain prerouting {
        type filter hook prerouting priority filter - 1; policy accept;
        ip daddr 172.168.0.40 ip daddr set 10.10.1.20 counter packets 1 bytes 84
    }
}
[root@localhost dit]# █
```

Figura 7.50 - Regla Router NAT escenario asimétrico

7. [Equipo:Router NAT - Usuario:root] Se abre tcpdump y se pone a escuchar en la interfaz “ens37”.

```
Equipo:Router NAT Usuario:root
```

```
$ tcpdump -i ens37
```

8. [Equipo:Encaminador - Usuario:root] Se abre el modo monitor de conntrack y el monitor de nftrace.

Además, se abre Wireshark y se pone a escuchar en la interfaz “ens39”

```
Equipo:Encaminador Usuario:root
```

```
#Terminal 1
$ conntrack -E --event-mask ALL

#Terminal 2
$ nft monitor trace
```

9. [Equipo:PC1 - Usuario:root] Se abre tcpdump y se pone a escuchar en la interfaz “ens37”.

```
Equipo:PC1 Usuario:root
```

```
$ tcpdump -i ens37
```

10. [Equipo:PC1 - Usuario:root] Desde PC1 se lanza un ping con dirección 172.168.0.20

```
Equipo:PC1 Usuario:root
```

```
$ ping -c 1 172.168.0.20
```

11. [Equipo:PC1 - Usuario:root] Observamos la captura de tcpdump de PC1.

En la Figura 7.51, podemos ver como el ping sale por la interfaz “ens37” con Ip destino 172.168.0.20

```

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens38, link-type EN10MB (Ethernet), capture size 262144 bytes
12:56:47.317429 IP 172.168.0.20 > localhost.localdomain: ICMP echo reply, id 15483, seq 1, length 64
12:56:47.360667 IP 8.8.8.8.domain > localhost.localdomain.50129: 43690 NXDomain 0/0/0 (42)

```

Figura 7.51 - Captura interfaz ens37 PC1 escenario asimétrico

12. [Equipo:Encaminador - Usuario:root] Observamos el monitor de nftrace de Encaminador.

En la Figura 7.52 se puede ver el camino realizado por el echo-request al llegar a Encaminador.

```

root@localhost:/home/dit#
root@localhost:/home/dit# nft monitor trace
trace id 149f276a netdev filteringress ingress packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether dad
dr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 172.168.0.20 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id
1773 ip length 84 icmp type echo-request icmp code 0 icmp id 16956 icmp sequence 1
trace id 149f276a netdev filteringress ingress rule nftrace set 1 (verdict continue)
trace id 149f276a netdev filteringress verdict continue
trace id 149f276a netdev filteringress ingress
trace id 149f276a ip filter prerouting verdict continue
trace id 149f276a ip filter prerouting
trace id 149f276a ip nat prerouting verdict continue
trace id 149f276a ip nat prerouting
trace id 149f276a ip filter forward verdict continue
trace id 149f276a ip filter forward
trace id 149f276a inet filter forward verdict continue
trace id 149f276a inet filter forward
trace id 149f276a ip filter postrouting verdict continue
trace id 149f276a ip filter postrouting
trace id 149f276a ip nat postrouting packet: oif "ens39" ip saddr 10.10.1.20 ip daddr 172.168.0.20 ip dscp
cs0 ip ecn not-ect ip ttl 63 ip id 1773 ip length 84 icmp type echo-request icmp code 0 icmp id 16956 icmp
sequence 1
trace id 149f276a ip nat postrouting rule ip saddr 10.10.1.20 counter packets 50 bytes 3991 snat to 172.16
8.0.40 (verdict accept)
trace id 54ea1505 netdev filterens39 ingress packet: iif "ens39" ether saddr 00:0c:29:91:4b:f2 ether daddr
00:0c:29:08:2e:2d ip saddr 172.168.0.20 ip daddr 172.168.0.40 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id
50833 ip length 84 icmp type echo-reply icmp code 0 icmp id 16956 icmp sequence 1
trace id 54ea1505 netdev filterens39 ingress rule nftrace set 1 (verdict continue)
trace id 54ea1505 netdev filterens39 ingress verdict continue
trace id 54ea1505 netdev filterens39 ingress

```

Figura 7.52 - Camino echo-request Encaminador en escenario asimétrico

En la Figura 7.52, se puede ver el camino del paquete ICMP enviado por PC1.

- 1) El paquete sale de PC1 y llega a Encaminador por la interfaz ens38
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena ingress de la tabla filteringress, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en la cadena ingress de filteringress es *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 4) En prerouting se procesa y se toma como veredicto *continue*.
- 5) Se consulta el encaminamiento del paquete, determinando que este ha de ser encaminado hacia el equipo PC2 (172.168.0.20).
- 6) Al tener que ser encaminado, el paquete continua su camino por el *hook forward*, siendo aquí su veredicto de *continue*.
- 7) El paquete llega el *hook postrouting*, donde hace match con la primera regla, realizando el cambio del campo Ip origen del paquete, en este caso la regla coloca como dirección origen del paquete 172.168.0.40, la cuál es la dirección Ip de la interfaz de Router NAT.
Con esto se consigue que cuando PC2 vaya a responder, enviará el paquete de respuesta a la ip 172.168.0.40 (Router NAT).
- 8) Se toma como veredicto *continue* y el paquete sale del equipo Encaminador por la interfaz “ens39” hacia PC2.

En la Figura 7.53, se puede ver la captura de Wireshark de la interfaz “ens39”, en la que se puede observar que se envía el ICMP request a PC2, teniendo este como dirección origen la Ip 172.168.0.40.

También podemos ver la respuesta que PC2 envía a Router NAT, teniendo dirección origen la Ip de PC2(172.168.0.20) e Ip destino la de Router NAT(172.168.0.40).

```
+ 7 46.664164669 172.168.0.40      172.168.0.20      ICMP      98 Echo (ping) request id=0x423c, seq=1/256, ttl=63 (reply in 8)
+ 8 46.726262031 172.168.0.20      172.168.0.40      ICMP      98 Echo (ping) reply id=0x423c, seq=1/256, ttl=64 (request in 7)
▶ Frame 7: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▶ Ethernet II, Src: Vmware_f0:01:28 (00:0c:29:f0:01:28), Dst: Vmware_91:4b:f2 (00:0c:29:91:4b:f2)
▶ Internet Protocol Version 4, Src: 172.168.0.40, Dst: 172.168.0.20
▼ Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x04ed [correct]
  [Checksum Status: Good]
  Identifier (BE): 16956 (0x423c)
  Identifier (LE): 15426 (0x3c42)
  Sequence number (BE): 1 (0x0001)
  Sequence number (LE): 256 (0x0100)
  [Response frame: 8]
  Timestamp from icmp data: Feb 2, 2021 22:55:46.000000000 CET
  [Timestamp from icmp data (relative): 7200.003492276 seconds]
  Data (48 bytes)
```

Figura 7.53 - Captura interfaz ens39 Encaminador en escenario asimétrico

13. [Equipo:Router NAT - Usuario:root] Observamos la captura de tcpdump del equipo Router NAT.

Cuando este recibe el paquete de PC2, se le aplica la regla de Stateless NAT y a través de la captura de TCPdump de su interfaz ens37 se puede ver como este encamina el paquete hacia PC1 (Figura 7.54).

```
[root@localhost dit]# tcpdump -i ens37
dropped privs to tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens37, link-type EN10MB (Ethernet), capture size 262144 bytes
13:01:09.919007 IP localhost.localdomain > 172.168.0.20: ICMP echo request, id 15483, seq 1, length 64
13:01:09.920540 IP localhost.localdomain.58487 > 8.8.8.8.domain: 2115+ PTR? 20.0.168.172.in-addr.arpa. (43)
13:01:14.925079 IP localhost.localdomain.58487 > 8.8.8.8.domain: 2115+ PTR? 20.0.168.172.in-addr.arpa. (43)
```

Figura 7.54 - Captura ens37 Router NAT escenario asimétrico

14. [Equipo:PC1 - Usuario:root] Observamos la captura de tcpdump del equipo PC1, así como la salida del comando ping ejecutado en el paso 10.

- En la Figura 7.55, se puede ver que el ping funciona y que la respuesta es capturada por la interfaz de PC1.

```
[root@localhost dit]# ping -c 1 172.168.0.20
PING 172.168.0.20 (172.168.0.20) 56(84) bytes of data.
64 bytes from 172.168.0.20: icmp_seq=1 ttl=63 time=45.1 ms

--- 172.168.0.20 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 45.062/45.062/45.062/0.000 ms
[root@localhost dit]#
```

Figura 7.55 - Captura ping PC1 escenario asimétrico

- En la Figura 7.56, se puede ver la captura de tcpdump, en la cual se captura un paquete con dirección origen 172.168.0.20 (PC2) y dirección destino PC1(10.10.1.20)

```

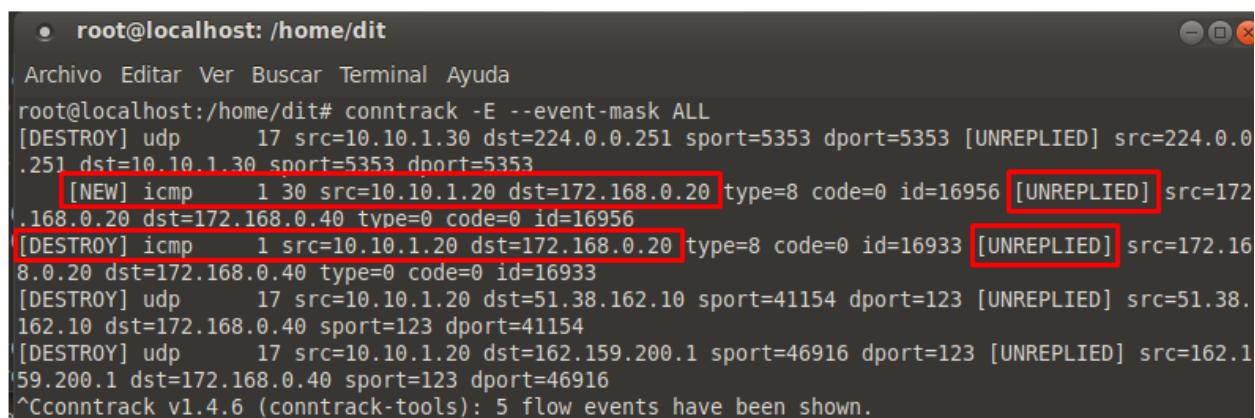
13:55:40.451580 IP 10.10.1.20 > 172.168.0.20: ICMP echo request, id 15483, seq 1, length 64
13:55:46.526193 IP 172.168.0.20 > 10.10.1.20: ICMP echo reply, id 15483, seq 1, length 64
13:55:51.674007 ARP, Request who-has 10.10.1.10 tell 10.10.1.20, length 46
13:55:51.674029 ARP, Reply 10.10.1.10 is-at 00:0c:29:f0:01:1e (oui Unknown), length 46
13:55:51.826732 ARP, Request who-has 10.10.1.20 tell localhost.localdomain, length 28
13:55:51.827392 ARP, Reply 10.10.1.20 is-at 00:0c:29:62:ed:ad (oui Unknown), length 46

```

Figura 7.56 - Captura ens37 PC1 echo reply escenario asimétrico

15. [Equipo:Encaminador - Usuario:root] Observamos la salida del monitor de eventos de conntrack de Encaminador.

Por último, queda ver que la tabla conntrack de Encaminador crea las entradas, pero en ningún momento puede colocarlas como assured o quitar el flag de UNREPLIED puesto que no ve tráfico de vuelta por este equipo (Figura 7.57).



```

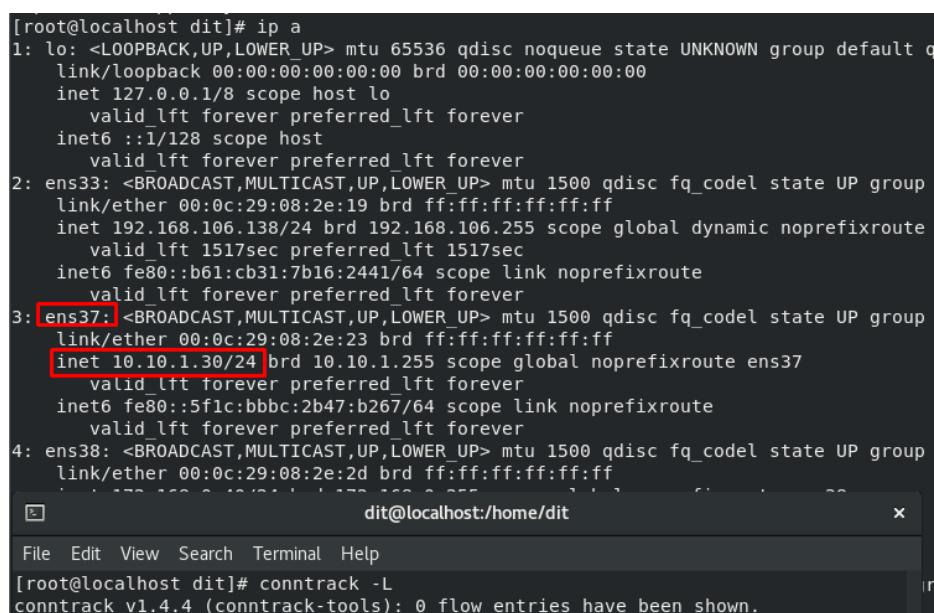
● root@localhost: /home/dit
Archivo Editar Ver Buscar Terminal Ayuda
root@localhost:/home/dit# conntrack -E --event-mask ALL
[DESTROY] udp      17 src=10.10.1.30 dst=224.0.0.251 sport=5353 [UNREPLIED] src=224.0.0.251 dst=10.10.1.30 sport=5353 dport=5353
[NEW]   icmp     1 30 src=10.10.1.20 dst=172.168.0.20 type=8 code=0 id=16956 [UNREPLIED] src=172.168.0.20 dst=172.168.0.40 type=0 code=0 id=16956
[DESTROY] icmp     1 src=10.10.1.20 dst=172.168.0.20 type=8 code=0 id=16933 [UNREPLIED] src=172.168.0.20 dst=172.168.0.40 type=0 code=0 id=16933
[DESTROY] udp      17 src=10.10.1.20 dst=51.38.162.10 sport=41154 dport=123 [UNREPLIED] src=51.38.162.10 dst=172.168.0.40 sport=123 dport=41154
[DESTROY] udp      17 src=10.10.1.20 dst=162.159.200.1 sport=46916 dport=123 [UNREPLIED] src=162.159.200.1 dst=172.168.0.40 sport=123 dport=46916
^Cconntrack v1.4.6 (conntrack-tools): 5 flow events have been shown.

```

Figura 7.57 - Entradas conntrack Encaminador escenario asimétrico

16. [Equipo:Encaminador - Usuario:root] Observamos las entradas de conntrack de Router NAT.

Router NAT tiene su tabla de conntrack vacía, puesto que se usa Stateless NAT (Figura 7.58).



```

[root@localhost dit]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inetc6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
    link/ether 00:0c:29:08:2e:19 brd ff:ff:ff:ff:ff:ff
    inet 192.168.106.138/24 brd 192.168.106.255 scope global dynamic noprefixroute
        valid_lft 1517sec preferred_lft 1517sec
    inetc6 fe80::b61:cb31:7b16:2441/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
3: ens37: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
    link/ether 00:0c:29:08:2e:23 brd ff:ff:ff:ff:ff:ff
    [REDACTED] 10.10.1.30/24 brd 10.10.1.255 scope global noprefixroute ens37
        valid_lft forever preferred_lft forever
    inetc6 fe80::5f1c:bbbc:2b26:7/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
4: ens38: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
    link/ether 00:0c:29:08:2e:2d brd ff:ff:ff:ff:ff:ff
        valid_lft forever preferred_lft forever
[root@localhost dit]# conntrack -L
conntrack v1.4.4 (conntrack-tools): 0 flow entries have been shown.

```

Figura 7.58 - Entradas conntrack equipo Router NAT escenario asimétrico

Conclusión del ejemplo:

En este ejemplo se ha podido comprobar el uso de los dos tipos de NAT que se tienen en el sistema (Stateful y Stateless).

Este ejemplo es el más importante de este capítulo, ya que comprende todos los conceptos que se han ido viendo a lo largo del mismo.

Como conclusiones destacar las siguientes:

No conviene realizar escenarios asimétricos, ya que como se ha podido comprobar el uso de los dos tipos de NAT es complicado y desaconsejable, puesto que se pierde la potencia del mismo.

En el equipo Encaminador, se puede ver que se realiza el Stateful NAT, pero que los paquetes en ningún momento ven respuesta de los mismos.

En el equipo Router NAT, se realiza el Stateless NAT, el uso del mismo está desaconsejado, por que como se ha podido comprobar, las respuestas no entran en las cadenas NAT, debido a que el *state* de estos es inválido.

El poder realizar el seguimiento de los paquetes en este tipo de escenarios es muy complejo, ya que todos los paquetes que lleguen a Encaminador de PC1, tendrán *state new* o *established*, mientras que los que pasen por Router Nat hacia PC1, tendrán todos *state invalid*, viendo que en este equipo puede usarse el módulo de conntrack para realizar el seguimiento de los flujos de comunicación.

Como punto final que resuma este capítulo, se desaconseja totalmente el uso de Stateless NAT, debido a la inutilidad del sistema de conntrack en estos casos, de forma que a la hora de plantear escenarios de red, estos tengan que ser Simétricos, pudiendo aprovechar así la potencia de conntrack para clasificar paquetes y que a estos puedan aplicárseles las reglas de traducción de direcciones por flujo de comunicación, en lugar de tener que realizar modificaciones (*mangle*) de los campos del paquetes, como si ocurría con Stateless NAT.

8 FILTRADO DE PAQUETES

En este capítulo, veremos las nuevas opciones que nftables ofrece con respecto a iptables, para el filtrado y tratamiento de los paquetes en la estructura de Netfilter.

Las opciones más destacables son; conjuntos dinámicos, mapas, ratios, quotas y contadores nombrados.

8.1 Conjuntos Dinámicos

Una de las novedades que ha implantado nftables con respecto a iptables, es la capacidad de poder manejar los conjuntos (sets [60]) de manera dinámica.

El set, es uno de los *Stateful objects*[61] que podemos tener en nftables, este puede ser de dos tipos;

- El set anónimo
- El set nominado

El set anónimo, es el que podemos crear en una regla para hacer una comprobación momentánea, pero el cuál no se puede manejar.

Un ejemplo de set anónimo puede ser la siguiente regla.

```
$ nft add rule ip filter input tcp dport {http, https} accept
```

El set creado, para poder decir si el puerto es 80 o 8080, es un set anónimo.

Como ya se ha mencionado, este set no puede ser manipulado, es decir, no se pueden añadir ni quitar elementos una vez creado.

Para poder manejar un set, necesitamos que este sea nominado.

El set nominado, es aquel que al crearlo se vincula a una tabla y en el cual se pueden añadir o quitar elementos de manera dinámica.

Un ejemplo de creación de set nominado, puede ser el siguiente comando (ver Figura 8.1).

```
$ nft add set filter 'direcciones { type ipv4_addr ; }'
```

Con este comando se creará un set, el cual albergará en su interior datos cuyo tipo serán direcciones Ipv4.

```
table ip filter {
    set direcciones {
        type ipv4_addr
    }
}
```

Figura 8.1 - Set dinámico

Para demostrar la potencia que puede tener el hecho de manejar un set de manera dinámica, se realizará un ejemplo de *port-knocking* [62].

8.1.1 Ejemplo de port-knocking

Descripción del ejemplo:

Este ejemplo servirá para ver la potencia de los sets nominados.

Para este ejemplo, se usará el escenario de 2 equipos, Anexo E - Escenario 2 Equipos.

- El equipo Encaminador, creará los sets y definirá las reglas en nftables, así como también monitorizará con *ntrace* el camino que estas siguen.
- El equipo PC1, tendrá que realizar una serie de peticiones a distintos puertos, en un orden determinado para que se le permita acceder al puerto 80, en el que estará escuchando el servidor Apache de Encaminador.

El orden de los puertos será; 100, 200 y finalmente el puerto 80 para poder comunicarse con el servidor.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Se creará una tabla filter que contendrá las cadenas; prerouting, input, forward, postrouting y output

```
Equipo:Encaminador Usuario:root

$ nft flush ruleset
$ nft add table ip filter
$ nft add chain ip filter prerouting '{ type filter hook prerouting priority 0 ; }'
$ nft add chain ip filter postrouting '{ type filter hook postrouting priority 0 ; }'
$ nft add chain ip filter input '{ type filter hook input priority 0 ; }'
$ nft add chain ip filter output '{ type filter hook output priority 0 ; }'
$ nft add chain ip filter forward '{ type filter hook forward priority 0 ; }'
```

2. **[Equipo:Encaminador - Usuario:root]** Se creará una tabla nat que contendrá las cadenas; pre y post.

```
Equipo:Encaminador Usuario:root

$ nft add table ip nat
$ nft add chain ip nat pre '{ type nat hook prerouting priority -10 ; }'
$ nft add chain ip nat post '{ type nat hook postrouting priority 10 ; }'
```

3. **[Equipo:Encaminador - Usuario:root]** Se creará una tabla ingress que contendrá la cadena ingr.

```
Equipo:Encaminador Usuario:root

$ nft add table netdev ingress
$ nft add chain netdev ingress ingr '{ type filter hook ingress device ens38 priority 0 ; }'
```

A continuación, se crean los sets y se definen las reglas, de forma que, para que un usuario pueda acceder al servicio Apache de Encaminador, tenga que realizar primero peticiones TCP al puerto 100, 200 , por ese orden, una vez lo haya realizado, se le permitirá la conexión con el servidor HTTP en el puerto 80.

Esto se conseguirá creando dos sets;

- El primer set, guardará la Ip de todos los equipos que hayan conseguido realizar correctamente el *port-knocking* y que por consiguiente tengan acceso al servidor HTTP.
- El segundo set, guardará la Ip de los equipos que intentan realizar el *port-knocking*

4. [Equipo:Encaminador - Usuario:root] Se procede a crear los sets, indicando que la información que guardarán será de tipo dirección Ipv4.

- El primer set guardará las direcciones de los equipos que tienen permitido el acceso al servidor HTTP.
- El segundo set guardará las direcciones de aquellos equipos que estén realizando el *port-knocking*.

```
Equipo:Encaminador Usuario:root

# Set 1
$ nft add set filter 'direcciones { type ipv4_addr ; }'

# Set 2
$ nft add set filter 'dir_portknock { type ipv4_addr ; }'
```

5. [Equipo:Encaminador - Usuario:root] Se crean las reglas del *port-knocking*.

- La primera regla, hará *match* si se produce una petición al puerto 100 con el protocolo TCP y se añadirá la Ip origen de este paquete, al set 2.
- La segunda regla, hará *match* si se produce una petición al puerto 200 con el protocolo TCP y se añadirá la Ip origen de este paquete, al set 1.
- La tercera regla, hará *match* si la dirección origen del paquete que quiere conectar con el servidor Apache se encuentra en el set 1.

```
Equipo:Encaminador Usuario:root

$ nft add rule ip filter input tcp dport 100 set add ip saddr @dir_portknock counter
$ nft add rule ip filter input tcp dport 200 ip saddr @dir_portknock
set add ip saddr @direcciones counter
$ nft add rule ip filter input tcp dport 80 ip saddr @direcciones
counter accept
$ nft add rule ip filter input ip protocol tcp drop
```

6. [Equipo:Encaminador - Usuario:root] Se crea una regla para poder realizar el seguimiento de los paquetes TCP que entran por la interfaz “ens38” de Encaminador.

```
Equipo:Encaminador Usuario:root

$ nft add rule netdev ingress ingr ip protocol tcp nftrace set 1
counter
```

7. [Equipo:Encaminador - Usuario:root] Se activa la monitorización de nftrace.

```
Equipo:Encaminador Usuario:root  
$ nft monitor trace
```

8. [Equipo:PC1 - Usuario:root] Se usa la herramienta hping3 para lanzar los paquetes TCP-SYN.

Desde PC1 y con la ayuda de la herramienta hping3, se procederá a lanzar los paquetes TCP SYN en el siguiente orden:

- El primero, se realizará al puerto 80 de Encaminador.
- El segundo, se realizará al puerto 100 de Encaminador.
- El tercero, se realizará al puerto 200 de Encaminador.
- El cuarto, se realizará de nuevo al puerto 80 de Encaminador, viendo como este último ya tiene acceso al servidor web.

```
Equipo:PC1 Usuario:root
```

```
$ hping3 -c 1 -S 10.10.1.10 -p 80  
$ hping3 -c 1 -S 10.10.1.10 -p 100  
$ hping3 -c 1 -S 10.10.1.10 -p 200  
$ hping3 -c 1 -S 10.10.1.10 -p 80
```

```
[root@localhost dit]# hping3 -c 1 -S 10.10.1.10 -p 80  
HPING 10.10.1.10 (ens37 10.10.1.10): S set, 40 headers + 0 data bytes  
--- 10.10.1.10 hping statistic ---  
1 packets transmitted, 0 packets received, 100% packet loss  
round-trip min/avg/max = 0.0/0.0/0.0 ms  
[root@localhost dit]# hping3 -c 1 -S 10.10.1.10 -p 100  
HPING 10.10.1.10 (ens37 10.10.1.10): S set, 40 headers + 0 data bytes  
--- 10.10.1.10 hping statistic ---  
1 packets transmitted, 0 packets received, 100% packet loss  
round-trip min/avg/max = 0.0/0.0/0.0 ms  
[root@localhost dit]# hping3 -c 1 -S 10.10.1.10 -p 200  
HPING 10.10.1.10 (ens37 10.10.1.10): S set, 40 headers + 0 data bytes  
--- 10.10.1.10 hping statistic ---  
1 packets transmitted, 0 packets received, 100% packet loss  
round-trip min/avg/max = 0.0/0.0/0.0 ms  
[root@localhost dit]# hping3 -c 1 -S 10.10.1.10 -p 80  
HPING 10.10.1.10 (ens37 10.10.1.10): S set, 40 headers + 0 data bytes  
len=46 ip=10.10.1.10 ttl=64 DF id=0 sport=80 flags=SA seq=0 win=64240 rtt=18.4 ms  
--- 10.10.1.10 hping statistic ---  
1 packets transmitted, 1 packets received 0% packet loss  
round-trip min/avg/max = 18.4/18.4/18.4 ms
```

Figura 8.2 - Secuencia de port-knocking

En la Figura 8.2, se puede observar cómo se ha realizado la llamada de puertos en el orden indicado, viendo como en el cuarto paquete, se obtiene el acceso al servidor Apache de Encaminador.

9. [Equipo:Encaminador - Usuario:root] Se procede a analizar las trazas de los paquetes, usando el monitor de *nfttrace*.

```
root@localhost:/home/dit# nft monitor trace
trace id e54b292b netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether d
addr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64
ip id 27431 ip length 40 tcp sport 1223 tcp dport http tcp flags == syn tcp window 512
trace id e54b292b netdev ingress ingr rule ip protocol tcp nftrace set 1 counter packets 0 byte
s 0 (verdict continue)
trace id e54b292b netdev ingress ingr verdict continue
trace id e54b292b netdev ingress ingr
trace id e54b292b ip filter prerouting verdict continue
trace id e54b292b ip filter prerouting
trace id e54b292b ip filter input packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr
00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64
ip id 27431 ip length 40 tcp sport 1223 tcp dport http tcp flags == syn tcp window 512
trace id e54b292b ip filter input rule ip protocol tcp drop (verdict drop)
```

Figura 8.3 - Primer intento de acceso al servidor web

En la Figura 8.3, se puede ver el camino de la primera petición realizada al puerto 80:

- 1) El paquete TCP SYN sale de PC1 y llega a Encaminador por la interfaz “ens38”
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena *ingr*, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en la cadena *ingr* es *continue*, por lo que el paquete continúa su camino hacia la cadena *prerouting* de *filter*.
- 4) Cruza el *hook prerouting*, siendo su veredicto *continue*.
- 5) Se consulta el encaminamiento del paquete, determinando que es para un proceso local (Apache), de forma que el siguiente *hook* al que se dirige es *input*.
- 6) En *input*, hace *match* con la cuarta regla, por lo que se hace *drop* del paquete.

```
trace id 4f5e3585 netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether d
addr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64
ip id 37651 ip length 40 tcp sport 2913 tcp dport 100 tcp flags == syn tcp window 512
trace id 4f5e3585 netdev ingress ingr rule ip protocol tcp nftrace set 1 counter packets 0 byte
s 0 (verdict continue)
trace id 4f5e3585 netdev ingress ingr verdict continue
trace id 4f5e3585 netdev ingress ingr
trace id 4f5e3585 ip filter prerouting verdict continue
trace id 4f5e3585 ip filter prerouting
trace id 4f5e3585 ip filter input packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr
00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64
ip id 37651 ip length 40 tcp sport 2913 tcp dport 100 tcp flags == syn tcp window 512
trace id 4f5e3585 ip filter input rule tcp dport 100 set add ip saddr @dir_portknock counter pa
ckets 0 bytes 0 (verdict continue)
trace id 4f5e3585 ip filter input rule ip protocol tcp drop (verdict drop)
```

Figura 8.4 - Primera secuencia del port knocking

En la Figura 8.4, se puede ver el camino de la segunda petición realizada al puerto 100:

- 1) El paquete TCP SYN sale de PC1 y llega a Encaminador por la interfaz “ens38”
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena *ingr*, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en la cadena *ingr* es *continue*, por lo que el paquete continúa su camino hacia la cadena *prerouting* de *filter*.

- 4) Cruza el *hook prerouting*, siendo su veredicto *continue*.
- 5) Se consulta el encaminamiento del paquete, determinando que es para un proceso local (Apache), de forma que el siguiente *hook* al que se dirige es *input*.
- 6) En *input*, hace *match* con la primera regla ya que su puerto destino es 100, de modo que se añade la Ip origen del paquete al set 2 y se toma como veredicto *continue*
- 7) Al seguir el procesado de las reglas, hace *match* con la cuarta regla, puesto que es tráfico TCP y se realiza el *drop* del paquete.

```
trace id 76bbbbf25 netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether d
addr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip tt
l 64 ip id 63208 ip length 40 tcp sport nessus tcp dport 200 tcp flags == syn tcp window 512
trace id 76bbbbf25 netdev ingress ingr rule ip protocol tcp nftrace set 1 counter packets 0 byte
s 0 (verdict continue)
trace id 76bbbbf25 netdev ingress ingr verdict continue
trace id 76bbbbf25 netdev ingress ingr
trace id 76bbbbf25 ip filter prerouting verdict continue
trace id 76bbbbf25 ip filter prerouting
trace id 76bbbbf25 ip filter input packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr
00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64
ip id 63208 ip length 40 tcp sport nessus tcp dport 200 tcp flags == syn tcp window 512
trace id 76bbbbf25 ip filter input rule tcp dport 200 ip saddr @dir_portknock set add ip saddr @
direcciones counter packets 0 bytes 0 (verdict continue)
trace id 76bbbbf25 ip filter input rule ip protocol tcp drop (verdict drop)
```

Figura 8.5 - Segunda secuencia del port-knocking

En la Figura 8.5, se puede ver el camino de la tercera petición realizada al puerto 200:

- 1) El paquete TCP SYN sale de PC1 y llega a Encaminador por la interfaz “ens38”
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena *ingr*, la cual activa la marca de seguimiento del paquete.
- 3) El veredicto en la cadena *ingr* es *continue*, por lo que el paquete continúa su camino hacia la cadena *prerouting* de *filter*.
- 4) Cruza el *hook prerouting*, siendo su veredicto *continue*.
- 5) Se consulta el encaminamiento del paquete, determinando que es para un proceso local (Apache), de forma que el siguiente *hook* al que se dirige es *input*.
- 6) En *input*, hace *match* con la segunda regla ya que su puerto destino es el 200 y la Ip origen de esta se encuentra registrada en el set 2, de modo que se añade la Ip origen del paquete al set 1 y se toma como veredicto *continue*.
- 7) Al seguir el procesado de las reglas, hace *match* con la cuarta regla, puesto que es tráfico TCP y se realiza el *drop* del paquete.

```
trace id 738efbc1 netdev ingress ingr verdict continue
trace id 738efbc1 netdev ingress ingr
trace id 738efbc1 ip filter prerouting verdict continue
trace id 738efbc1 ip filter prerouting
trace id 738efbc1 ip filter input packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr
00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64
ip id 46292 ip length 40 tcp sport 2845 tcp dport http tcp flags == syn tcp window 512
trace id 738efbc1 ip filter input rule tcp dport http ip saddr @direcciones counter packets 0 b
ytes 0 accept (verdict accept)
```

Figura 8.6 - Petición al servidor aceptada después del port-knocking

En la Figura 8.6, se puede ver el camino de la primera petición realizada al puerto 80:

- 1) El paquete TCP SYN sale de PC1 y llega a Encaminador por la interfaz “ens38”
 - 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena `ingr`, la cual activa la marca de seguimiento del paquete.
 - 3) El veredicto en la cadena `ingr` es *continue*, por lo que el paquete continúa su camino hacia la cadena `prerouting` de filter.
 - 4) Cruza el *hook prerouting*, siendo su veredicto *continue*.
 - 5) Se consulta el encaminamiento del paquete, determinando que es para un proceso local (Apache), de forma que el siguiente *hook* al que se dirige es `input`.
 - 6) En `input`, hace *match* con la tercera regla ya que el paquete es TCP y su dirección se encuentra registrada en el set 1, por lo que se toma como veredicto *accept*.
10. [Equipo:PC1 - Usuario:root] Una vez terminado el proceso, se procede a observar que los set's han sido actualizados y que aparece la Ip de PC1 en ambos, tanto en el set del intento de llamada como en el set de direcciones aceptadas.

Se puede observar la salida de los comandos en la Figura 8.7.

Equipo: Encaminador **Usuario:**root

```
# Este set guarda las direcciones de los que están realizando
# portknocking
$ nft list set filter dir_portknock

# Este set guarda las direcciones que se aceptan para acceder al
# servidor Apache
$ nft list set filter direcciones
```

```
root@localhost:/home/dit# nft list set filter dir_portknock
table ip filter {
    set dir_portknock {
        type ipv4 addr
        elements = { 10.10.1.20 }
    }
}
root@localhost:/home/dit# nft list set filter direcciones
table ip filter {
    set direcciones {
        type ipv4 addr
        elements = { 10.10.1.20 }
    }
}
root@localhost:/home/dit#
```

Figura 8.7 - Set's actualizados dinámicamente

En la Figura 8.8, se puede observar un resumen del camino de los paquetes en el ejemplo de *port-knocking*

En color rojo se puede ver el camino de **TCP- SYN puerto 100**
 En color naranja se puede ver el camino de **TCP-SYN puerto 200**
 En color morado se puede ver el camino de **TCP-SYN puerto 80**
 En color azul se puede ver el camino de **TCP SYN-ACK**

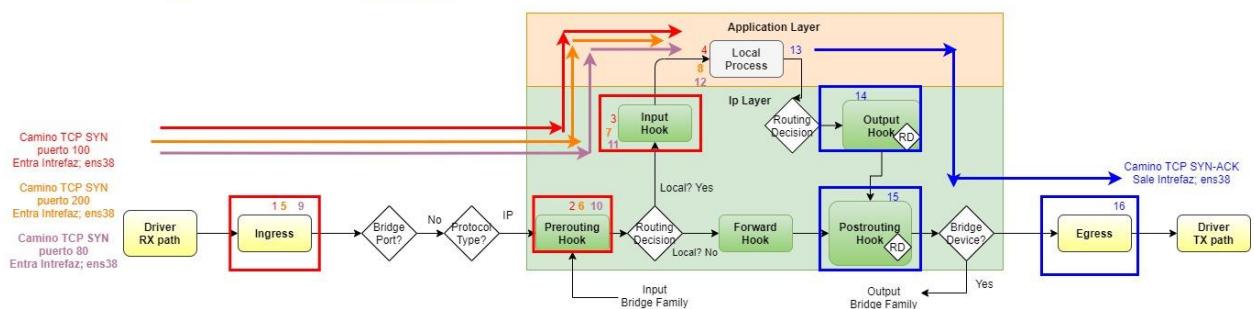


Figura 8.8 - Resumen camino paquetes ejemplo port-knocking

Conclusión del ejemplo:

Como se ha podido ver en el ejemplo, el uso de los set's es novedoso en la herramienta de nftables.

Estos set's, nos proveen de una manera dinámica al realizar el tratamiento de un paquete, haciendo por ejemplo un *port-knocking*.

Un administrador de redes, puede utilizar los sets para implementar pequeñas pruebas de IDS, registrando los datos de los paquetes que tengan un comportamiento sospechoso para impedir el acceso de los mismos.

Esta ventaja por ejemplo, permitiría realizar la conexión de Ip's que provengan de una red interna, a las cuales se les realiza una serie de comprobaciones (cabeceras mal formadas, numero de paquetes por segundo, tipo de paquetes etc...) y en caso de que alguna de estas comprobaciones sea positiva, se podría registrar la Ip del equipo que tiene este tipo de comportamiento y limitarle la conexión.

También se pueden crear set's en los cuales, los elementos permanezcan en su interior un tiempo determinado.

```
$ nft add set filter 'llamando { type ipv4_addr ; timeout 1m ; }'
```

En este ejemplo, el elemento no permanecerá en el interior más de un minuto, de esta manera se puede realizar el mismo ejemplo que en el caso anterior, pero sabiendo que el orden de llamada en los puertos debe hacerse en un tiempo determinado, en caso contrario la Ip almacenada en el set se eliminará.

8.2 Mapas Dinámicos

Los mapas [63], son un tipo de estructura que han sido añadidas en nftables.

Los mapas permiten realizar clasificaciones de tráfico mucho más detalladas de las que se podían realizar en iptables.

Un mapa, es una estructura que se basa en una pareja de datos:

- El primer campo, hace referencia al tipo de dato que se desea comprobar (direcciones ip, tipos de protocolos, marca de conexiones etc...)
- El segundo campo, hace referencia a la acción que se ha de tomar ante una coincidencia con el primer tipo de dato.

Se realizarán dos pruebas:

- En la primera, se realizará una detección temprana de Ip's
- En la segunda, se realizará una división de tráfico según el protocolo del mismo.

8.2.1 Ejemplo mapa direcciones Ip

Descripción del ejemplo:

En este ejemplo, se usará un mapa para permitir o denegar el tráfico en función de la dirección destino del paquete.

Se realizará una prueba con dos direcciones destinos, la primera dirección será la de PC2, la cual será rechazada y la segunda dirección será la de Encaminador, la cual aceptaremos.

Para poder realizar este ejemplo, se partirá de la configuración realizada en el “Ejemplo de port-knocking”, el cual usa la configuración del escenario de 2 equipos “Anexo E - Escenario 2 Equipos”.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Se crea el mapa y se añade el elemento con el que este trabajará.

Antes de crear el mapa, se limpian de reglas las cadenas; input, postrouting y forward de la tabla filter y la cadena ingr de la tabla ingress.

- La primera regla, crea un mapa llamado mydict, el cual guardará datos del tipo ipv4.
- La segunda regla, añade un elemento al mapa, este elemento asocia direcciones ipv4 con acciones terminales.

```
Equipo:Encaminador Usuario:root

#Limpieza de cadenas
$ nft flush chain ip filter postrouting
$ nft flush chain ip filter forward
$ nft flush chain ip filter input
$ nft flush chain netdev ingress ingr

#Reglas
$ nft add map filter mydict { type ipv4_addr : verdict\; }
$ nft add element filter mydict { 10.10.1.10 : accept, 172.168.0.20
: drop }
```

Se ha creado un mapa, en el cual el dato clave que se va a usar es de tipo dirección Ipv4, este dato lleva asociada una acción o veredicto.

Se añaden elementos a ese mapa, de forma que se acepte la dirección Ip 10.10.1.10 y se rechace la 172.168.0.20, ahora bien, debe quedar claro que se están definiendo direcciones Ip, en ningún momento se indica si son origen o destino, eso dependerá del uso que se le dé al mapa.

En este ejemplo, se usarán como direcciones destino.

2. [Equipo:Encaminador - Usuario:root] Se asocia el mapa creado a una cadena, para ello se usa una regla que lo contenga, en este caso, se añade una regla a la cadena prerouting que compruebe si la dirección destino es la del equipo Encaminador o la del equipo PC2.

Se activa también en la cadena ingr de la tabla ingress la marca *nftrace*, para aquellos paquetes cuyo protocolo sea ICMP.

Equipo: Encaminador Usuario: root

```
$ nft add rule filter prerouting ip saddr vmap @mydict
$ nft add rule netdev ingress ingr ip protocol icmp nftrace set 1
counter
```

3. [Equipo:Encaminador - Usuario:root] Se activa el monitor de *nftrace* para poder ver el camino que siguen los paquetes dentro del *ruleset* de nftables.

Equipo: Encaminador Usuario: root

```
$ nft monitor trace
```

4. [Equipo:PC1 - Usuario:dit] Desde PC1, se realiza un ping a la dirección de PC2 y otro a la dirección de Encaminador.

Equipo: PC1 Usuario: dit

```
$ ping -c 1 172.168.0.20
$ ping -c 1 10.10.1.10
```

En la Figura 8.9, se puede observar como el ping realizado a 172.168.0.20 no obtiene respuesta, mientras que el realizado a 10.10.1.10 si la tiene.

```
[dit@localhost ~]$ ping -c 1 172.168.0.20
PING 172.168.0.20 (172.168.0.20) 56(84) bytes of data.

--- 172.168.0.20 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

[dit@localhost ~]$ ping -c 1 10.10.1.10
PING 10.10.1.10 (10.10.1.10) 56(84) bytes of data.
64 bytes from 10.10.1.10: icmp_seq=1 ttl=64 time=0.333 ms

--- 10.10.1.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.333/0.333/0.333/0.000 ms
[dit@localhost ~]$
```

Figura 8.9 - Ping PC1 ejemplo mapa

5. [Equipo:Encaminador - Usuario:root] Se procede a comprobar la salida que ofrece el monitor *nftrace* del transcurso de los paquetes.

Se puede ver la salida del monitor *nftrace* en la Figura 8.10.

```

root@localhost:/home/dit# nft monitor trace
trace id d42fec7a netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether d
addr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 172.168.0.20 ip dscp cs0 ip ecn not-ect ip
ttl 64 ip id 7946 ip length 84 icmp type echo-request icmp code 0 icmp id 4490 icmp sequence 1
trace id d42fec7a netdev ingress ingr rule.ip protocol icmp nftrace set 1 counter packets 0 byt
es 0 (verdict continue)
trace id d42fec7a netdev ingress ingr verdict continue
trace id d42fec7a netdev ingress ingr
trace id d42fec7a ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether
daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 172.168.0.20 ip dscp cs0 ip ecn not-ect ip
ttl 64 ip id 7946 ip length 84 icmp type echo-request icmp code 0 icmp id 4490 icmp sequence 1
trace id d42fec7a ip filter prerouting rule.ip daddr vmap @mydict (verdict drop)
trace id 34c088e6 netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether d
addr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip tt
l 64 ip id 2402 ip length 84 icmp type echo-request icmp code 0 icmp id 4505 icmp sequence 1
trace id 34c088e6 netdev ingress ingr rule.ip protocol icmp nftrace set 1 counter packets 0 byt
es 0 (verdict continue)
trace id 34c088e6 netdev ingress ingr verdict continue
trace id 34c088e6 netdev ingress ingr
trace id 34c088e6 ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether
daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip t
tl 64 ip id 2402 ip length 84 icmp type echo-request icmp code 0 icmp id 4505 icmp sequence 1
trace id 34c088e6 ip filter prerouting rule.ip daddr vmap @mydict (verdict accept)
trace id 34c088e6 ip filter input verdict continue
trace id 34c088e6 ip filter input
^C
root@localhost:/home/dit# 

```

Figura 8.10 - Camino paquetes ejemplo mapa

En la Figura 8.10, se puede ver el camino de los dos pings realizados por PC1:

- Ping hacia **PC2**:
 - 1) El paquete ICMP con destino a PC2 (172.168.0.20), sale de PC1 y llega a Encaminador por la interfaz “ens38”.
 - 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena `ingr`, la cual activa la marca de seguimiento del paquete.
 - 3) El veredicto en la cadena `ingr` es *continue*, por lo que el paquete continúa su camino hacia la cadena `prerouting` de filter.
 - 4) En `prerouting`, hace *match* con la regla que contiene el mapa creado, en este coincide con el segundo elemento guardado, el cual determina que con dirección Ip destino PC2, la acción a tomar es *drop*.
 - 5) Se hace *drop* del paquete como resultado de la acción determinada por el mapa.

- Ping hacia **Encaminador**:
 - 1) El paquete ICMP con destino a Encaminador (10.10.1.10) sale de PC1 y llega a Encaminador por la interfaz “ens38”.
 - 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena `ingr`, la cual activa la marca de seguimiento del paquete.
 - 3) El veredicto en la cadena `ingr` es *continue*, por lo que el paquete continúa su camino hacia la cadena `prerouting` de filter.
 - 4) En `prerouting`, hace *match* con la regla que contiene el mapa creado, en este coincide con el primer elemento guardado, el cual determina que con dirección Ip destino Encaminador, la acción a tomar es *accept*.
 - 5) Se toma la decisión de routing sobre el paquete y viendo que este es para un proceso local, continúa su camino por `input`.
 - 6) En `input`, se toma como decisión del paquete *continue*, entregándose al proceso local.

En la Figura 8.11, se puede ver un resumen del camino seguido por los paquetes en este ejemplo.

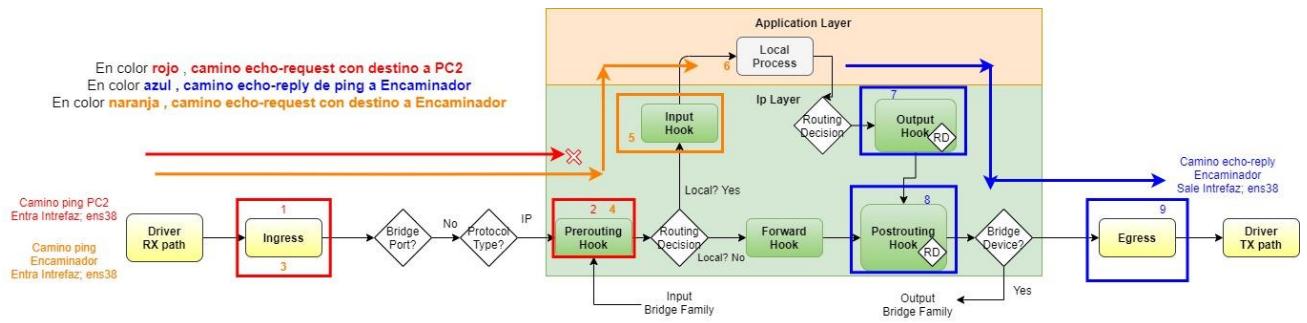


Figura 8.11 - Camino ejemplo mapa direcciones Ip's

8.2.2 Ejemplo mapa protocolos

Descripción del ejemplo:

Para realizar este ejemplo, se creará un mapa, el cuál albergará datos de tipo protocolos y cuyos veredictos serán acciones *jump* a las cadenas de usuario específicas de cada protocolos.

En cada cadena de usuario, se puede dar un tratamiento específico a los paquetes de esos protocolos, bien puede ser; marcarlos, realizarles un *drop* o *reject* temprano, llevar la contabilidad de los mismos etc...

En este ejemplo, se usará la base de tablas y cadenas creada en el “Ejemplo mapa direcciones Ip”, la cual usa el escenario de 2 equipos del “Anexo E - Escenario 2 Equipos”.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Partiendo de la estructura base vista en el ejemplo anterior, se introducen los siguientes comandos.

En primer lugar, se limpia de reglas la cadena prerouting de la tabla filter.

- 1) Se crea un mapa, el cual tendrá la relación de datos protocolos-veredicto.
- 2) Se crean 3 cadenas de usuario, una por cada tipo de protocolo (TCP, UDP e ICMP).
- 3) Se añade al mapa un elemento por cada protocolo, indicándole como acción el *jump* a su cadena de usuario correspondiente.

```
Equipo:Encaminador Usuario:root
```

```
#Limpiamos las reglas del ejemplo anterior
$ nft flush chain ip filter prerouting

#Creacion del mapa
$ nft add map filter protocolos { type inet_proto : verdict\; }

#Creación de las cadenas de usuario
$ nft add chain ip filter cadena-tcp
$ nft add chain ip filter cadena-udp
$ nft add chain ip filter cadena-icmp
```

```
#Se añaden los elementos al mapa creado
$ nft add element ip filter protocolos { tcp : jump cadena-tcp, }
$ nft add element ip filter protocolos { icmp : jump Icmp, }
$ nft add element ip filter protocolos { udp : jump cadena-udp, }
```

2. [Equipo:Encaminador - Usuario:root] A continuación, se añadirán las reglas de tratamiento según el protocolo;

- A los paquetes ICMP, se les realizará la contabilidad, se marcará su flujo con un 0x2 y aquellos que tengan como destino una red externa, se les hará un *reject*.
- Los paquetes UDP serán rechazados.
- Los paquetes TCP, se aceptarán si su *state* es *New* o *Established* y se realizará la contabilidad de los mismos.
- Además, se añadirá el mapa como regla a la cadena prerouting de filter.

Equipo:Encaminador **Usuario:**root

```
#Reglas ICMP
$ nft add rule ip filter Icmp ip saddr 10.10.1.20 ct mark set 0x2
continue
$ nft add rule ip filter Icmp ip daddr != 10.10.1.10 reject with
icmp type host-unreachable
$ nft add rule ip filter Icmp counter

#Reglas UDP
$ nft add rule ip filter cadena-udp counter drop

#Reglas TCP
$ nft add rule ip filter cadena-tcp ct state { new, established }
counter accept
$ nft add rule ip filter cadena-tcp counter drop
$ nft add rule ip filter prerouting ip protocol vmap @protocolos
```

3. [Equipo:Encaminador - Usuario:root] Se comprueba la estructura final de cadenas y reglas (ver Figura 8.12).

Además de ello, se inicia un nuevo terminal, en el cual se abre el monitor de *nftrace*.

Equipo:Encaminador **Usuario:**root

```
$ nft list ruleset

#Terminal 2
$ nft monitor trace
```

```

table ip filter {
    map protocolos {
        type inet_proto : verdict
        elements = { icmp : jump cadena-icmp, tcp : jump cadena-tcp, udp : jump cadena-udp }
    }

    map mydict {
        type ipv4_addr : verdict
    }

    chain prerouting {
        type filter hook prerouting priority 0; policy accept;
    }

    chain postrouting {
        type filter hook postrouting priority 0; policy accept;
    }

    chain input {
        type filter hook input priority 0; policy accept;
    }

    chain output {
        type filter hook output priority 0; policy accept;
    }

    chain forward {
        type filter hook forward priority 0; policy accept;
    }

    chain cadena-tcp {
        ct state { established, new } counter packets 0 bytes 0 accept
        counter packets 0 bytes 0 drop
    }

    chain cadena-udp {
        counter packets 0 bytes 0 drop
    }

    chain cadena-icmp {
        ip saddr 10.10.1.20 ct mark set 0x00000002 continue
        ip daddr != 10.10.1.10 reject with icmp type host-unreachable
        counter packets 0 bytes 0
    }
}

```

Figura 8.12 - Estructura reglas ejemplo mapa protocolos

4. [Equipo:PC1 - Usuario:root] Se procede a realizar las pruebas generando tráfico desde PC1.

Equipo: PC1 **Usuario:**root

```

#Tráfico ICMP
$ ping 10.10.1.10

#Tráfico UDP
$ nslookup 8.8.8.8

#Tráfico TCP
$ hping3 -R 10.10.1.10

```

5. [Equipo:Encaminador - Usuario:root] Se comprueba si las cadenas de usuario han procesado el tráfico antes de pasar a comprobar el camino de los paquetes.

Equipo: Encaminador **Usuario:**root

```
$ nft list ruleset
```

```

        chain cadena-tcp {
            ct state { established, new } counter packets 2 bytes 80 accept
            counter packets 1 bytes 40 drop
        }

        chain cadena-udp {
            counter packets 214 bytes 16323 drop
        }

        chain Icmp {
            ip saddr 10.10.1.20 ct mark set 0x00000002 continue
            ip daddr != 10.10.1.10 reject with icmp type host-unreachable
            counter packets 1 bytes 84
        }
    }
}

```

Figura 8.13 - Cadenas del mapa después de procesar tráfico

Como se puede comprobar en la Figura 8.13, las reglas han contado tráfico y en las figuras sucesivas se puede ver el camino de ICMP, UDP y TCP respectivamente, viendo que en todas ellas, cuando el paquete llega al *hook prerouting*, se le aplica la regla vmap para que según el protocolo del paquete que llegue, salte a una u otra cadena.

6. [Equipo:Encaminador - Usuario:root] Se comprueba con la ayuda del monitor de *ntrace* el camino seguido por cada tipo de paquete.

```

root@localhost:/home/dit#
root@localhost:/home/dit# nft monitor trace
trace id 40f91f23 netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:6
2:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip
dscp cs0 ip ecn not-ect ip ttl 64 ip id 31867 ip length 84 icmp type echo-req
st icmp code 0 icmp id 3755 icmp sequence 1
trace id 40f91f23 netdev ingress ingr rule ip saddr 10.10.1.20 nftrace set 1 cou
nter packets 611 bytes 45108 (verdict continue)
trace id 40f91f23 netdev ingress ingr verdict continue
trace id 40f91f23 netdev ingress ingr
trace id 40f91f23 ip nat pre verdict continue
trace id 40f91f23 ip nat pre
trace id 40f91f23 ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:
62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 i
p dscp cs0 ip ecn not-ect ip ttl 64 ip id 31867 ip length 84 icmp type echo-req
est icmp code 0 icmp id 3755 icmp sequence 1
trace id 40f91f23 ip filter prerouting rule ip protocol vmap @protocolos (verdic
t jump Icmp)
trace id 40f91f23 ip filter Icmp rule ip saddr 10.10.1.20 ct mark set 0x00000002
continue (verdict continue)
trace id 40f91f23 ip filter Icmp rule counter packets 0 bytes 0 (verdict continu
e)

```

Figura 8.14 - Camino paquete ICMP ejemplo mapa protocolos

En la Figura 8.14, se puede ver el camino del ping realizado por PC1:

- 1) El paquete ICMP con destino a Encaminador (10.10.1.10) sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena *ingr*, la cual activa la marca de seguimiento del paquete.
- 3) En la cadena *ingr* se activa su marca *nftrace* y se toma como veredicto *continue*, por lo que el paquete continúa su camino hacia la cadena *prerouting* de *filter*.
- 4) En *prerouting*, hace *match* con la regla que contiene el mapa creado y el elemento del mapa determina que el protocolo ICMP, debe saltar a la cadena de usuario “Icmp”.
- 5) En la cadena de usuario, hace *match* con la primera y la tercera regla, marcando el flujo del

paquete con un 2 y aumenta el contador de la tercera regla.

- 6) El paquete termina su procesamiento en la cadena de usuario y vuelve a la cadena prerouting, donde su veredicto es *continue*
- 7) Se toma la decisión de routing sobre el paquete y al estar dirigido a un proceso local, continúa su camino por el *hook input*.
- 8) En *input*, se toma como veredicto *continue* y se pasa al proceso local.

```
trace id 481b7737 ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 8.8.8.8 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 18023 ip length 79 udp sport 56986 udp dport domain udp length 59
trace id 481b7737 ip filter prerouting rule ip protocol vmap @protocolos (verdict jump cadena-udp)
trace id 481b7737 ip filter cadena-udp rule counter packets 159 bytes 12388 drop
; (verdict drop)
```

Figura 8.15 - Camino de paquete UDP ejemplo mapa protocolos

En la Figura 8.15, se puede ver el camino de la consulta al DNS realizada por PC1:

- 1) El paquete UDP con destino a la máquina pública (8.8.8.8) sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena *ingr*, la cual activa la marca de seguimiento del paquete.
- 3) En la cadena *ingr* se activa su marca *nftrace* y se toma como veredicto *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de *filter*.
- 4) En prerouting, hace *match* con la regla que contiene el mapa creado y el elemento del mapa determina que el protocolo UDP, debe saltar a la cadena de usuario “cadena-udp”.
- 5) En la cadena de usuario, hace *match* con la primera regla, tomando como veredicto el *drop* del paquete y terminando así el procesado de este.

```
trace id 6ba17d21 ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 64 ip id 0 ip length 40 tcp sport 2395 tcp dport http tcp flags == rst tcp window 0
trace id 6ba17d21 ip filter prerouting rule ip protocol vmap @protocolos (verdict jump cadena-tcp)
trace id 6ba17d21 ip filter cadena-tcp rule ct state { } counter packets 0 bytes 0 accept (verdict accept)
trace id 6ba17d21 ip filter input verdict continue
trace id 6ba17d21 ip filter input
```

Figura 8.16 - Camino de paquete TCP ejemplo mapa protocolos

En la Figura 8.16, se puede ver el camino del TCP-RST realizado por PC1:

- 1) El paquete TCP con destino a Encaminador (10.10.1.10) sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena *ingr*, la cual activa la marca de seguimiento del paquete.
- 3) En la cadena *ingr*, se activa su marca *nftrace* y se toma como veredicto *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de *filter*.
- 4) En prerouting, hace *match* con la regla que contiene el mapa creado y el elemento del mapa

determina que protocolo TCP, debe saltar a la cadena de usuario “cadena-tcp”.

- 5) En la cadena de usuario, hace *match* con la primera regla, al tener este paquete el estado ESTABLISHED se le da como acción *accept*.
- 6) Se toma la decisión de *routing* sobre el paquete y al estar dirigido a un proceso local, continúa su camino por el *hook input*.
- 7) En *input*, se toma como veredicto *continue* y se pasa al proceso local.

En la Figura 8.17, se puede ver un esquema del camino seguido por los paquetes de este ejemplo.

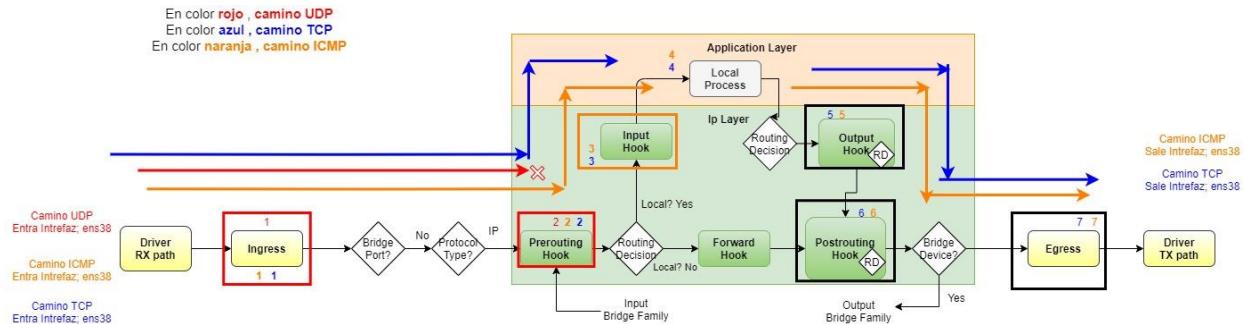


Figura 8.17 - Camino ejemplo mapa direcciones protocolos

Conclusión del ejemplo:

En este ejemplo se ha podido ver el uso de los mapas en nftables.

Al igual, que se vio en el capítulo 3 “Cadena de usuario”, la potencia de las cadenas de usuario para dividir el tráfico, el mapa es el complemento perfecto para este tipo de cadenas.

Los mapas permiten realizar una división del tráfico más precisa, ordenada y focalizada, de la que se realizaría mediante reglas.

Otro punto a favor del uso de mapas, es la opción de poder modificarlos de forma dinámica, consiguiendo que la clasificación del tráfico puede modificarse según las necesidades de la red o del administrador.

Es por ello que en los sistemas reales, esta opción es ampliamente usada en conjunto con las cadenas de usuario, permitiendo la diferenciación del tráfico, no solo por protocolos, sino por una gran variedad de opciones, gracias a los tipos de elementos de los mapas.

8.3 Medidas de tráfico

En este punto se verán varias opciones usadas por nftables para trabajar con el tráfico que llega hasta él, bien sea limitándolo, creando contadores nombrados o mediante el uso de quotas.

Una vez se haya visto estos tres métodos para la inspección de tráfico, se llevará a cabo un ejemplo que nos permitirá usar las medidas de tráfico en conjunto con los sets dinámicos, como es una lista negra.

8.3.1 Limitación de ratio

En nftables, se pueden establecer reglas que limiten el tráfico que supere un cierto nivel de paquetes/bytes dado,

o que por el contrario, no se procese hasta llegar a cierto volumen de datos.

Esto se consigue usando la opción *limit rate*.

Con esta opción, se puede indicar si se desea que la limitación sea, bien por (*packets*) o por (*bytes*).

En la Figura 8.18, se puede observar que se han creado 4 reglas;

- La primera de ellas, realiza *match* con aquellos flujos que se encuentren por debajo del límite de paquetes por segundo que se ha establecido.
- La segunda de ellas por el contrario, realiza *match* si el flujo de datos supera el número de paquetes por segundo dado.
- La tercera regla trabaja con bytes en lugar de paquetes, haciendo *match* a aquellos flujos que estén por debajo de los bytes/segundos que se han establecido.
- La cuarta regla trabaja también con bytes, haciendo *match* a aquellos flujos que superen el límite establecido.

Equipo: Encaminador **Usuario:** root

```
$ nft add rule filter input ip saddr . ip protocol { 10.10.1.20 . tcp}
counter limit rate over 10/second drop
$ nft add rule filter input ip saddr . ip protocol { 10.10.1.20 . tcp}
counter limit rate over 10/second drop
$ nft add rule filter input ip saddr . ip protocol { 10.10.1.20 . tcp}
counter limit rate 1000bytes/second accept
$ nft add rule filter input ip saddr . ip protocol { 10.10.1.20 . tcp}
counter limit rate 1 mbytes/second drop
```

```
chain input {
    type filter hook input priority 0; policy accept;
    ip saddr . ip protocol { 10.10.1.20 . tcp } counter packets 23 bytes 920 limit rate 10/second burst 5 packets accept
    ip saddr . ip protocol { 10.10.1.20 . tcp } counter packets 0 bytes 0 limit rate over 10/second burst 5 packets drop
    ip saddr . ip protocol { 10.10.1.20 . tcp } counter packets 0 bytes 0 limit rate 1000 bytes/second accept
    ip saddr . ip protocol { 10.10.1.20 . tcp } counter packets 0 bytes 0 limit rate 1 mbytes/second drop
}
```

Figura 8.18 - Reglas limit rate

Además del uso de *limit rate*, se puede observar que se ha hecho uso de una nueva funcionalidad que implementa nftables, que es la concatenación.

Mediante el uso de la concatenación, se puede exigir que varios campos de un paquete coincidan con los datos que se proporcionen en la regla.

En este caso se ha usado la siguiente regla:

```
ip saddr . ip protocol { 10.10.1.20 . tcp}
```

Con esto, estamos indicando que al paquete que llegue a esta regla, se le revisaran dos campos concatenados, la Ip origen y el protocolo y que estos coincidan con los datos que se le han pasado a la regla, en este caso son; la dirección Ip origen debe ser 10.10.1.20 y que el protocolo del paquete con esa dirección origen sea TCP.

Las concatenaciones, al igual que las reglas, se deben leer de izquierda a derecha y siguen el mismo comportamiento que un AND lógico. Para que se acepte el *match*, todas las concatenaciones previas deben cumplirse dentro del mismo paquete.

Además, se puede observar que en las dos primera líneas viene por defecto la opción *burst*.

Esta opción, permite exceder el límite de tráfico puesto en el número de paquetes/bytes que esta indique.

En el caso de la regla que se ha creado, implica que en un pico de tráfico puede superar la limitación de 10 paquetes por segundo en 5 paquetes más, hasta tener un total de 15 paquetes por segundo.

Esta opción se establece por defecto para las limitaciones de paquetes, no así a las limitaciones de datos.

8.3.2 Quotas

Una quota es un *stateful object* usado por nftables, que se parece mucho a *limit rate*.

Como se ha visto en el punto anterior, *limit rate* se aplica al *match* en una regla y es independiente según el flujo de datos, es decir, cada flujo de datos contabiliza su propio límite, sin tener en cuenta el resto de flujos posibles.

No ocurre así con la quota, la cuál contabiliza el tráfico total de todos los flujos que realicen *match* con esa regla.

8.3.2.1 Ejemplo quota

Descripción del ejemplo:

Se realizará un ejemplo rápido del uso del objeto quota, partiendo de la estructura creada en el “Ejemplo de port-knocking” y usando el escenario de 2 equipos “Anexo E - Escenario 2 Equipos”.

- En Encaminador, se creará una quota del protocolo TCP, a la cuál se le dará un tamaño de 10 kbytes.
- Desde PC1, se realizará un escaneo de nmap con la opción -sS que generará paquetes TCP con el flag SYN.

Se podrá observar que tras llegar al límite de esa quota, se procede a realizar el *drop* de los paquetes.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** En primer lugar, se crea una quota de 10000 bytes y se añade una regla que la use dentro de la cadena input de filter.

```
Equipo:Encaminador Usuario:root
$ nft add quota ip filter 'quota-tcp over 10000 bytes'
$ nft add rule ip filter input ip protocol tcp quota name quota-tcp counter drop
```

2. **[Equipo:Encaminador - Usuario:root]** Se comprueba la quota creada, mostrando el *ruleset* de nftables.

En la Figura 8.19 se puede observar la quota creada.

```
Equipo:Encaminador Usuario:root
$ nft list ruleset / $ nft list quota ip filter
```

```

quota quota-tcp {
    over 10000 bytes
}

map protocolos {
    type inet_proto : verdict
    elements = { icmp : jump Icmp, tcp : jump cadena-tcp, udp : jump cadena-udp }
}

map mydict {
    type ipv4_addr : verdict
}

chain prerouting {
    type filter hook prerouting priority 0; policy accept;
}

chain postrouting {
    type filter hook postrouting priority 0; policy accept;
}

chain input {
    type filter hook input priority 0; policy accept;
    ip protocol tcp quota name "quota-tcp" counter packets 0 bytes 0 drop
}

```

Figura 8.19 - Ejemplo quota

3. [Equipo:Encaminador - Usuario:root] Se procede a activar la monitorización de *nfttrace* en Encaminador.

Equipo:Encaminador Usuario:root

```
$ nft monitor trace
```

4. [Equipo:PC1 - Usuario:root] En PC1, se usa el siguiente comando para comenzar con la prueba del escenario.

Nmap enviará paquetes TCP-SYN a la dirección dada (Encaminador).

Equipo:PC1 Usuario:root

```
$ nmap -sS 10.10.1.10
```

5. [Equipo:Encaminador - Usuario:root] Tras completarse el envío de nmap, se puede comprobar en la Figura 8.20, que la quota se ha cumplido y que ha aceptado paquetes hasta llegar al límite dado (10 kbytes) y luego se ha realizado el *drop* del resto.

```

quota quota-tcp {
    over 10000 bytes used 10000 bytes
}

map protocolos {
    type inet_proto : verdict
    elements = { icmp : jump Icmp, tcp : jump cadena-tcp, udp : jump cadena-udp }
}

map mydict {
    type ipv4_addr : verdict
}

chain prerouting {
    type filter hook prerouting priority 0; policy accept;
}

chain postrouting {
    type filter hook postrouting priority 0; policy accept;
}

chain input {
    type filter hook input priority 0; policy accept;
    ip protocol tcp quota name "quota-tcp" counter packets 3456 bytes 152064 drop
}

```

Figura 8.20 - Quota cumplida y paquetes con acción drop

En la Figura 8.21, se puede comprobar como los primeros paquetes que llegaban actualizaban la quota, pero en ningún momento realizaban *match* con la regla de input, de forma que a estos no se les realizaba el *drop*.

6. [Equipo:Encaminador - Usuario:root] Se comprueba el monitor de *nftrace* para ver el camino de los paquetes que eran aceptados y de los que han sido rechazados por la quota.

Si se visualiza la Figura 8.21 y la Figura 8.22, se puede ver como tras cumplirse la quota especificada para TCP, los paquetes realizan *match* con la regla y se les aplica la acción *drop*.

```
trace id b2fa8a35 netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 54 ip id 12127 ip length 44 tcp sport 45470 tcp dport ssh tcp flags == syn tcp window 1024
trace id b2fa8a35 netdev ingress ingr rule ip saddr 10.10.1.20 nftrace set 1 counter packets 4176 bytes 210773 (verdict continue)
trace id b2fa8a35 netdev ingress ingr verdict continue
trace id b2fa8a35 netdev ingress ingr
trace id b2fa8a35 ip nat pre verdict continue
trace id b2fa8a35 ip nat pre
trace id b2fa8a35 ip filter prerouting verdict continue
trace id b2fa8a35 ip filter prerouting
trace id b2fa8a35 ip filter input verdict continue
trace id b2fa8a35 ip filter input
```

Figura 8.21 - Paquetes antes de cumplir la quota

En la Figura 8.21, se puede ver el camino del TCP-SYN realizado por PC1:

- 1) El paquete TCP con destino a Encaminador (10.10.1.10) sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena *ingr*, la cual activa la marca de seguimiento del paquete.
- 3) En la cadena *ingr* se activa su marca *nftrace* y se toma como veredicto *continue*, por lo que el paquete continúa su camino hacia la cadena *prerouting* de *filter*.
- 4) En *prerouting* actualiza la quota y se toma como veredicto *continue*.
- 5) Se toma la decisión de *routing* sobre el paquete, al estar dirigido a un proceso local, este continúa su camino por el *hook input*.
- 6) En *input*, se toma como veredicto *continue* y se le pasa al proceso local.

```
trace id b13fb4ee netdev ingress ingr rule ip saddr 10.10.1.20 nftrace set 1 counter packets 4176 bytes 210773 (verdict continue)
trace id b13fb4ee netdev ingress ingr verdict continue
trace id b13fb4ee netdev ingress ingr
trace id b13fb4ee ip nat pre verdict continue
trace id b13fb4ee ip nat pre
trace id b13fb4ee ip filter prerouting verdict continue
trace id b13fb4ee ip filter prerouting
trace id b13fb4ee ip filter input packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether daddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl 57 ip id 65181 ip length 44 tcp sport 45470 tcp dport radm-port tcp flags == syn tcp window 1024
trace id b13fb4ee ip filter input rule ip protocol tcp quota name "quota-tcp" counter packets 1932 bytes 85008 drop (verdict drop)
```

Figura 8.22 - Paquetes después de cumplir la quota

En la Figura 8.22, se puede ver el camino del TCP-SYN realizado por PC1:

- 1) El paquete TCP con destino a Encaminador(10.10.1.10) sale de PC1 y llega a Encaminador por

la interfaz “ens38”.

- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena ingr, la cual activa la marca de seguimiento del paquete.
- 3) En la cadena ingr se activa su marca *nftrace* y se toma como veredicto *continue*, por lo que el paquete continúa su camino hacia la cadena prerouting de filter.
- 4) En prerouting, hace *match* con la regla de la quota, ya que esta ha sido superada, por lo que se toma como acción el *drop* del paquete.

Conclusión del ejemplo:

Como se ha podido ver en el ejemplo, la quota es la opción más viable, para controlar el volumen de tráfico de la red.

Esta opción resulta muy interesante para un administrador de red, ya que puede limitar el volumen de tráfico de ciertos servicios, o usar esta opción para implementar平衡adores de carga que eviten la saturación de la red.

También es buena opción el uso de quotas, para establecer límites de conectividad según el tipo de usuarios que se tengan en una red, ampliándole el volumen a ciertos clientes y recortando el de otros.

8.3.3 Contadores nombrados

Hasta ahora, se ha visto como crear contadores en cada regla, pero si en lugar de querer un contador por regla, se busca tener un contador que pueda ser usado en diferentes cadenas dentro de una misma tabla, se ha de crear lo que se conoce como contador nombrado.

Este tipo de contadores, es otro tipo de *stateful object* dentro de nfatbles.

Al crearse este tipo de contadores, se les asigna un nombre y se pueden usar desde distintas reglas/cadenas haciendo referencia al nombre de este.

8.3.3.1 Ejemplo contadores nombrados

Descripción del ejemplo:

Se creará un contador nombrado dentro de la tabla filter y este se colocará en dos cadenas distintas para comprobar como el tráfico que pase por estas, lo incrementa.

De forma que el resultado mostrado, será el cómputo total de los paquetes que han atravesado ambas cadenas

Para realizar este ejemplo, se partirá de la estructura creada en el “Ejemplo de port-knocking”, el cual se basa en el escenario de 2 equipos “Anexo E - Escenario 2 Equipos”.

Análisis del ejemplo:

1. **[Equipo:Encaminador - Usuario:root]** Se crea el contador nombrado y las reglas que lo usen del siguiente modo.
 - 1) Se crea el contador con el nombre contador-ping.
 - 2) Se añade el contador a la cadena prerouting.
 - 3) Se añade el contador a la cadena output.

Equipo:Encaminador **Usuario:**root

```
#Se crea el contador con el nombre contador-ping
$ nft add counter ip filter contador-ping

#Se añade el contador a la cadena output
$ nft add rule ip filter output ip protocol icmp 'counter name
contador-ping counter accept'

#Se añade el contador a la cadena prerouting
$ nft add rule ip filter prerouting ip protocol icmp 'counter name
contador-ping counter accept'
```

2. **[Equipo:Encaminador - Usuario:root]** Se comprueba que el contador y las reglas se han creado correctamente.

Equipo:Encaminador **Usuario:**root

```
$ nft list ruleset
```

En la Figura 8.23, se puede ver como se ha creado el contador y las reglas dentro de la estructura de nftables.

```
counter contador-ping {
    packets 0 bytes 0
}

chain prerouting {
    type filter hook prerouting priority 0; policy accept;
    ip protocol icmp counter name "contador-ping" counter packets 0 bytes 0 accept
}

chain postrouting {
    type filter hook postrouting priority 0; policy accept;
}

chain input {
    type filter hook input priority 0; policy accept;
}

chain output {
    type filter hook output priority 0; policy accept;
    ip protocol icmp counter name "contador-ping" counter packets 0 bytes 0 accept
}
```

Figura 8.23 - Estructura contador nombrado y reglas

3. **[Equipo:PC1 - Usuario:dit]** Desde PC1 se realiza un ping hacia el equipo Encaminador.

Equipo:PC1 **Usuario:**dit

```
$ ping -c 3 10.10.1.10
```

4. **[Equipo:Encaminador - Usuario:root]** Se puede comprobar en la Figura 8.24, que el contador nombrado que se ha creado, ha aumentado por cada paquete ICMP que ha cruzado tanto por la cadena de prerouting como por la cadena de output.

Equipo: Encaminador **Usuario:** root

```
$ nft list ruleset
```

```
counter contador-ping {  
    packets 6 bytes 504  
}  
  
chain prerouting {  
    type filter hook prerouting priority 0; policy accept;  
    ip protocol icmp counter name "contador-ping" counter packets 3 bytes 252 accept  
}  
  
chain postrouting {  
    type filter hook postrouting priority 0; policy accept;  
}  
  
chain input {  
    type filter hook input priority 0; policy accept;  
}  
  
chain output {  
    type filter hook output priority 0; policy accept;  
    ip protocol icmp counter name "contador-ping" counter packets 3 bytes 252 accept  
}
```

Figura 8.24 - Prueba ejemplo contador nombrado

8.4 Ejemplo lista negra

Descripción del ejemplo:

En este punto, se realizará un ejemplo de cómo se puede crear una lista negra, aplicando los conceptos de filtrado de paquetes de nftables.

Para ello, se partirá de la estructura usada en el “Ejemplo de port-knocking”, la cual usa el escenario de 2 equipos “Anexo E - Escenario 2 Equipos”.

En este ejemplo, se buscará poner de manifiesto la potencia del nuevo *framework* de Netfilter en comparación a iptables.

Se creará un set dinámico, el cual tendrá datos de tipo `ipv4_address` y se le asignará un `timeout` de 1 hora, a los elementos de este set. En este set se almacenará las Ip's de la lista negra.

Para realizar esta prueba, PC1 usará la herramienta nmap para realizar un análisis de puertos, enviando paquetes TCP vacíos con el flag SYN activado.

Para evitar que este escaneo pueda llevarse a cabo, en el equipo Encaminador se creará una regla de filtrado que inspeccionará los paquetes y que establecerá un límite de paquetes TCP/seg que tengan activo el flag SYN.

Una vez superado este límite de paquetes, se añadirá la dirección Ip origen del paquete al set dinámico creado.

Análisis del ejemplo:

1. [Equipo:Encaminador - Usuario:root] Se crean las siguientes reglas en Encaminador.
 - Se crea un set llamado “listanegra”, en el cual se guardarán las direcciones que deseamos bloquear.
 - La primera regla que se añade a la cadena prerouting, es para que compruebe si la dirección origen del paquete se encuentra en el set “listanegra” y en ese caso, realice el *drop* del mismo.
 - La segunda regla, añade una limitación de tráfico TCP con la flag SYN a un límite de 10

paquetes por segundo, con una opción de *burst* de 5 paquetes y en caso de cumplir esto, se añade la Ip origen del paquete al set “listanegra”.

- Por seguridad, se añade una regla en output, que realice el *drop* de los paquetes cuya dirección destino se encuentre en el set “listanegra”.

Equipo:Encaminador **Usuario:**root

```
$ nft add set ip filter' listanegra { type ipv4_addr ; timeout 1h ; }'  
$ nft add rule ip filter prerouting ip saddr @listanegra counter drop  
$ nft add rule ip filter prerouting tcp flags syn limit rate over 10/second set add ip saddr @listanegra drop  
$ nft add rule ip filter output ip daddr @listanegra drop
```

2. **[Equipo:Encaminador - Usuario:root]** Se activa la monitorización de reglas con *nftrace* en Encaminador y se abre un terminal Wireshark, para que escuche en la interfaz “ens38”.

Equipo:Encaminador **Usuario:**root

```
$ nft monitor trace
```

3. **[Equipo:PC1 - Usuario:root]** En PC1, se usa el siguiente comando para realizar un escaneo de puertos TCP-SYN y así poner a prueba las reglas de Encaminador.

Equipo:PC1 **Usuario:**root

```
$ nmap -SS 10.10.1.10
```

4. **[Equipo:Encaminador - Usuario:root]** Se comprueba el camino de los paquetes en el monitor de *nftrace*.

Si se observa la Figura 8.25, se puede ver el paso de los paquetes por el sistema de nftables y que la regla actúa rápidamente añadiendo la Ip de PC1 a la lista negra.

En la imagen, además se puede apreciar el momento en el cual atraviesa el último paquete antes de sobrepasar el límite establecido, siendo este capaz de acceder al sistema, teniendo un veredicto de *accept* en la regla de la cadena input.

Mientras que el siguiente paquete, ya realiza *match* con la regla creada en Encaminador y se le aplica el veredicto *drop*.

```

trace id 58afa272 netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether da
ddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl
39 ip id 33287 ip length 44 tcp sport 38033 tcp dport telnet.tcp flags == syn tcp window 1024
trace id 58afa272 netdev ingress ingr rule ip protocol tcp nftrace set 1 counter packets 6517 by
tes 299852 (verdict continue)
trace id 58afa272 netdev ingress ingr verdict continue
trace id 58afa272 netdev ingress ingr
trace id 58afa272 ip filter prerouting verdict continue
trace id 58afa272 ip filter prerouting
trace id 58afa272 ip filter input verdict continue
trace id 58afa272 ip filter input
trace id 0bbdac9 netdev ingress ingr packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether da
ddr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl
46 ip id 15929 ip length 44 tcp sport 38033 tcp dport pop3.tcp flags == syn tcp window 1024
trace id 0bbdac9 netdev ingress ingr rule ip protocol tcp nftrace set 1 counter packets 6517 by
tes 299852 (verdict continue)
trace id 0bbdac9 netdev ingress ingr verdict continue
trace id 0bbdac9 netdev ingress ingr
trace id 0bbdac9 ip filter prerouting packet: iif "ens38" ether saddr 00:0c:29:62:ed:ad ether d
addr 00:0c:29:f0:01:1e ip saddr 10.10.1.20 ip daddr 10.10.1.10 ip dscp cs0 ip ecn not-ect ip ttl
46 ip id 15929 ip length 44 tcp sport 38033 tcp dport pop3.tcp flags == syn tcp window 1024
trace id 0bbdac9 ip filter prerouting rule tcp.flags syn limit rate over 10/second burst 5 pack
ets set add ip saddr @listanegra drop (verdict drop)

```

Figura 8.25 - Camino paquetes lista negra

En la Figura 8.25, se puede ver el camino de los paquetes TCP-SYN admitidos, enviados por PC1:

- 1) El paquete TCP con destino a Encaminador (10.10.1.10) sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena *ingr*, la cual activa la marca de seguimiento del paquete.
- 3) En la cadena *ingr*, se activa su marca *nftrace* y se toma como veredicto *continue*, por lo que el paquete continúa su camino hacia la cadena *prerouting* de *filter*.
- 4) En *prerouting*, se toma como veredicto *continue*.
- 5) Se toma la decisión de *routing* sobre el paquete, al estar dirigido a un proceso local, este continúa su camino por el *hook input*.
- 6) En *input*, se toma como veredicto *continue* y se le pasa al proceso local.

En la Figura 8.25, también se puede comprobar el punto a partir del cual, se añade la Ip de PC1 al set “*listanegra*”, viendo como se realiza el *drop* del resto de paquetes.

- 1) El paquete TCP con destino a Encaminador (10.10.1.10) sale de PC1 y llega a Encaminador por la interfaz “ens38”.
- 2) Tras realizar las comprobaciones básicas del mismo, la NIC le pasa el paquete a la cadena *ingr*, la cual activa la marca de seguimiento del paquete.
- 3) En la cadena *ingr* se activa su marca *nftrace* y se toma como veredicto *continue*, por lo que el paquete continúa su camino hacia la cadena *prerouting* de *filter*.
- 4) En *prerouting* hace *match* con la regla de limitación del tráfico, ya que se habrán enviado más de 10 paquetes TCP-SYN por segundo, de forma que esta regla añade la Ip origen del paquete al set “*listanegra*” y realiza el *drop* de este.
- 5) El resto de paquetes TCP con dirección origen PC1 que entren en el sistema, harán *match* en *prerouting* con la primera regla, la cual realizará el *drop* de los paquetes.

5. [Equipo:Encaminador - Usuario:root] Se comprueba la captura de tráfico realizada por Wireshark.

En la Figura 8.26, se puede apreciar una captura de tráfico de Wireshark, en la cual se muestran que los paquetes enviados por PC1 no consiguen respuesta ninguna por parte de Encaminador.

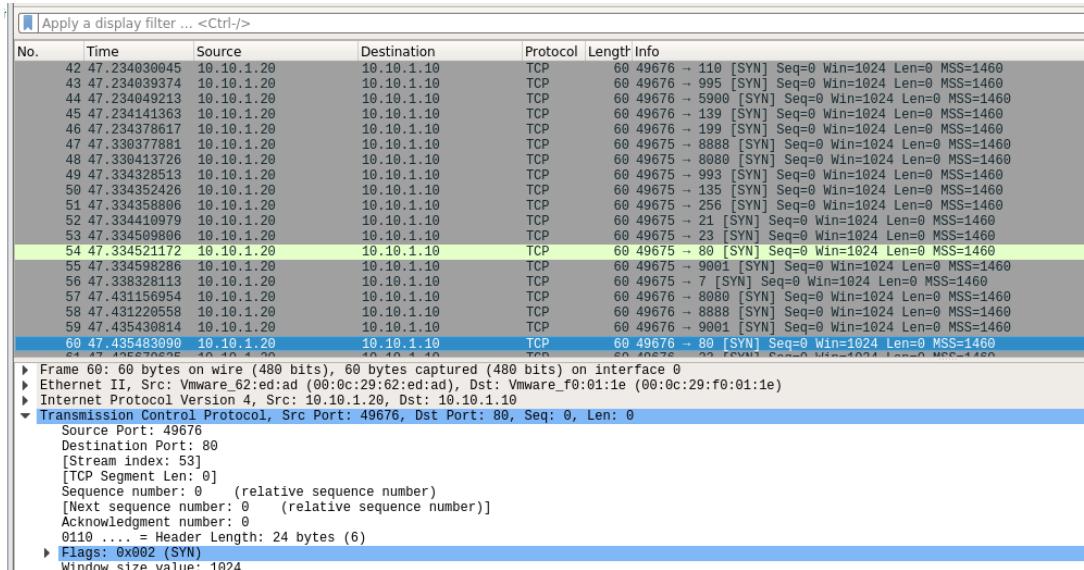


Figura 8.26 - Captura de tráfico ejemplo lista negra

6. [Equipo:Encaminador - Usuario:root] Se observa el *ruleset* de nftables, para comprobar que la Ip de PC1 se encuentra guardada en el set y que el temporizador de 1 hora que se establece para la Ip, está disminuyendo (ver Figura 8.27).

Además de ello, se puede comprobar el contador de la regla que cuenta el número de intentos de acceso de una Ip que se encuentra en el set “listanegra”.

```
root@localhost:/home/dit# nft list ruleset
table ip filter {
    map mydict {
        type ipv4_addr : verdict
        elements = { 10.10.1.10 : accept, 172.168.0.20 : drop }
    }

    set listanegra {
        type ipv4_addr
        timeout 1h
        elements = { 10.10.1.20 expires 49m40s }
    }

    chain prerouting {
        type filter hook prerouting priority 0; policy accept;
        ip saddr @listanegra counter packets 1728 bytes 76040 drop
        tcp flags syn limit rate over 10/second burst 5 packets set add ip saddr @listanegra drop
        tcp flags syn limit rate over 3/second burst 5 packets set add ip saddr @listanegra drop
        tcp flags syn limit rate over 3/second burst 5 packets set add ip saddr @listanegra drop
    }
}
```

Figura 8.27 - Reglas y contador de lista negra

7. [Equipo:PC1 - Usuario:root] Por último, se comprueba el resultado que arroja nmap en PC1, el cual, tal y como se puede observar en la Figura 8.28, indica que los puertos se encuentran filtrados por algún *firewall*.

```
[root@localhost dit]# nmap -sS 10.10.1.10
Starting Nmap 7.70 ( https://nmap.org ) at 2021-02-07 11:52 PST
Nmap scan report for 10.10.1.10
Host is up (0.00026s latency).
All 1000 scanned ports on 10.10.1.10 are filtered
MAC Address: 00:0C:29:F0:01:1E (VMware)

Nmap done: 1 IP address (1 host up) scanned in 34.73 seconds
[root@localhost dit]#
```

Figura 8.28 - Resultado nmap ejemplo lista negra

Conclusión del ejemplo:

En este ejemplo se ha podido comprobar un buen ejemplo de uso de las opciones de filtrado de nftables.

La implementación realizada con los sets y los mapas, que permiten que estos se modifiquen de manera dinámica, permite realizar configuraciones imposibles en iptables.

Se ha visto como realizar una lista negra, añadiendo la Ip de un equipo con comportamiento malicioso a un set dinámico que impida su conectividad durante un periodo de tiempo.

Un administrador de redes, también podría querer implementar listas blancas, que podría modificar de forma dinámica según el comportamiento inspeccionado de los usuarios.

Otro ejemplo podría ser realizar configuraciones en las cadenas que permitieran el bastionado de los servicios, haciéndose este proceso de forma dinámica.

Las nuevas funcionalidades de nftables, permiten gestiones mucho más dinámicas de las permitidas en iptables y aunque estas opciones no estén totalmente desarrolladas y sigan añadiéndoles funcionalidades, ya se puede comprobar el potencial de las mismas y los nuevos usos que pueden tener.

9 CONCLUSIONES

En este capítulo se presentarán las conclusiones e ideas obtenidas tras la realización del proyecto e investigar su documentación y su implementación.

Como idea general se puede comprobar que existe una gran reticencia a cambiar de iptables a nftables, ya que la persona que se ha acostumbrado a trabajar con el lenguaje de iptables le resulta cómodo y no encuentra grandes complicaciones en este, a esto se le ha de sumar, que numerosos managers y gestores han atrasado su migración o están comenzado a realizarla ahora, por lo que tampoco ofrece un extra de seguridad.

Antes de avanzar con este capítulo, he de decir, que la principal dificultad ha sido , la de la información, ya que es escasa y la que se ha conseguido encontrar , no está revisada y existen numerosas inexactitudes con respecto al manejo y procesos de la misma.

9.1 Objetivos conseguidos

En el siguiente punto se mostrarán los resultados conseguidos, del estudio realizado de nftables:

- Uno de los principales objetivos del proyecto era conseguir una guía de usuario consolidada, la cual se ha conseguido de una manera bastante completa, ya que no solo explica el manejo de la herramienta, si no que todo lo realizado se encuentra ejemplificado y detallado con imágenes de las salidas, comandos y estructuras usadas.

El punto de partida del uso de nftables viene detallado en el punto 3, “Nftables Introducción”

- Otro punto importante a estudiar en la guía era el camino seguido por los paquetes, indicando que *hooks* y en qué orden se atraviesan, debido a que la mayoría de información que se encuentra en la red, tiene inexactitudes y plantea caminos que no son del todo ciertos.

Se ha conseguido detallar con claridad el camino que sigue un paquete dentro de Netfilter, diviendo su transcurso según el tipo de paquete, indicando el camino paso a paso que este realiza por los diferentes *hooks*.

Este punto puede verse bien desarrollado en el punto 4, “Flujo de Paquetes (Packet Flow)”.

- Además de ello se ha realizado un ejemplo que explica y detalla visualmente , el transcurso de los paquetes, en el camino de la interfaz “lo”, ya que acerca de este camino, pueden encontrarse por la red numerosas propuestas, las cuales no son del todo ciertas.

Este ejemplo queda detallado en el punto “Escenario prueba flow interfaz lo”

- Uno de los puntos más importantes desarrollados dentro de esta guía, ha sido el estudio del módulo conntrack.

Aunque este es un módulo de Netfilter y no es propiamente de nftables, apenas existe información de su funcionamiento y de las entradas de seguimiento que este crea y con las que opera.

Se ha conseguido realizar un análisis de las entradas que este crea con los protocolos TCP, UDP e ICMP, realizando un estudio de los campos que las componen y los posibles valores de las mismas, cubriendo así la carencia del estudio de estas, ya que no se puede encontrar información acerca de las mismas. Estas se pueden ver en los siguientes puntos:

- “Entrada conntrack TCP”
- “Entrada conntrack UDP”
- “Entrada conntrack ICMP”

- Otro punto importante dentro del estudio de conntrack, ha sido el realizado acerca del funcionamiento de los helpers, ya que la única información que podía encontrarse de los mismos, era su finalidad.

En el punto “Helpers” se ha realizado un análisis del funcionamiento de los mismos, mostrando el estado de los paquetes y la creación de las entradas en conntrack.

Del mismo modo, se ha analizado las entradas de la tabla *expect* de conntrack, de la cual no hay ni documentación, ni tampoco explicación de los campos que conforman estas entradas.

Gracias a la ayuda de Pablo Neira , el cual ha trabajado y colaborado en del desarrollo del módulo conntrack, se han podido aclarar las dudas del mismo , así como detallar los procesos que esta sigue y usar ejemplo precisos para ver el comportamiento del mismo.

- El análisis del módulo NAT también ha resultado complejo de analizar.

Se ha conseguido en el proyecto, explicar paso a paso, que es y como funciona la traducción de direcciones, y el uso de esta dentro de nftables “Traducción de direcciones (NAT)”.

La explicación de NAT, se ha basado en la creación de escenarios de pruebas, detallando tanto su funcionalidad, como su uso y el montaje de los mismos, permitiendo que cualquier persona que siga esta guía pueda replicarlos y comprobar por si misma los resultados obtenidos del estudio.

- Otro de los Objetivos principales de este proyecto era el desarrollo de una guía, que pudiese publicarse y que sirviera tanto a nivel teórico como a nivel práctico.
- Este proyecto busca también colaborar con la wiki de la herramienta, la cual se completará con los capítulos de este trabajo una vez presentado.

El objetivo principal es mostrar el potencial de esta nueva herramienta, si bien es verdad que hay funcionalidades de iptables, que todavía no han sido añadidas a nftables, pero también es cierto, que esta cuenta con mejoras novedosas como pueden ser la inclusión de su nuevo hook ingress, de la implementación de los set y los diccionarios y de una mejora en la trazabilidad de los paquetes a través de las reglas y cadenas.

9.2 Futuras líneas de investigación

Nftables es un proyecto que se encuentra vivo y que sigue mejorando e implantando nuevas y mejores soluciones para facilitar la migración, así como por ejemplo se plantea la inclusión de otro nuevo tipo de hook llamado egress, el cual iría después del hook postrouting.

Así también se podría plantear un cambio en conntrack, que en lugar de ser un mero espectador y de mantener el estado del paquete que atraviesa por él, fuera capaz de llevar una dualidad de estados entre en cliente y el servidor, de forma que se pueda ver en qué punto de la comunicación se encuentra cada uno, ya que estos estados no tienen por qué coincidir siempre.

Otra línea de trabajo para el futuro son los laboratorios de prácticas, ya que la comunidad ha creado numerosos escenarios de prácticas en los cuales probar la herramienta de iptables, pero apenas existen escenarios de nftables.

Un hándicap visto en números sitios es la falta de documentación, de la misma, o las inexactitudes en la información que los usuarios pueden encontrar por la red. Pero esto es un problema que se está subsanando, completando la wiki de la herramienta y ofreciendo más detalles acerca de las diferentes opciones de uso de esta herramienta.

Visto esto, solo queda esperar y ver si la reticencia que existe ahora mismo en la comunidad, en cuanto a la migración de una herramienta a otra, va desapareciendo y con la ayuda de los managers y el apoyo de las distintas distribuciones de linux, se consiga la transición.

ANEXO A - TECNOLOGÍAS USADAS Y HERRAMIENTAS

En este anexo se detallan las tecnologías que se han usado en el desarrollo de la guía.

VMWare

Es la herramienta usada para virtualizar los sistemas usados a la hora de plantear los escenarios y crear los distintos equipos usados para la realización de las pruebas.

Para crear una máquina virtual en VMWare se siguen los siguientes pasos:

1º Descargar las imágenes de los S.O que se desean usar, en el caso de esta guía, una máquina Debian y una CentOS.

2º Creamos la virtualización del sistema operativo en VMWare.

- 1) Se pulsa en File > New Virtual Machine... (Figura A.0.1)

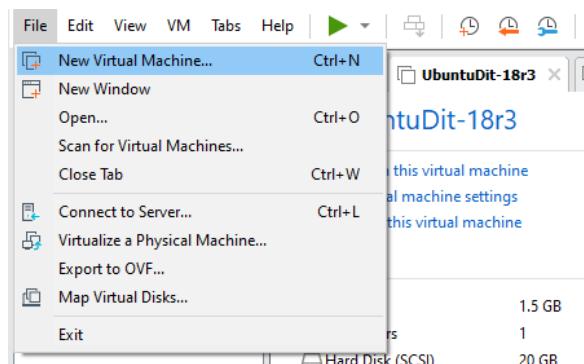


Figura A.0.1 - Crear máquina virtual

- 2) Se usa la opción *Installer disc image file* (iso) y se le da la ruta donde se tenga guardada la imagen iso de los sistemas descargados (Figura A.0.2).

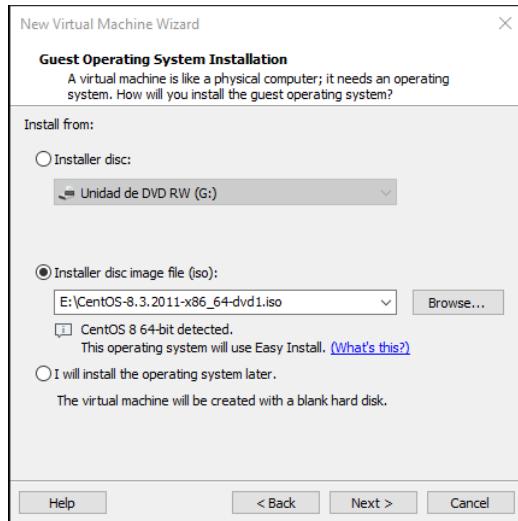


Figura A.0.2 - Cargar imagen (.iso) máquina virtual

- 3) Se establece el nombre del usuario y su contraseña, hay que tener en cuenta, que la contraseña establecida será también la usada por el superusuario *root* (Figura A.0.3).

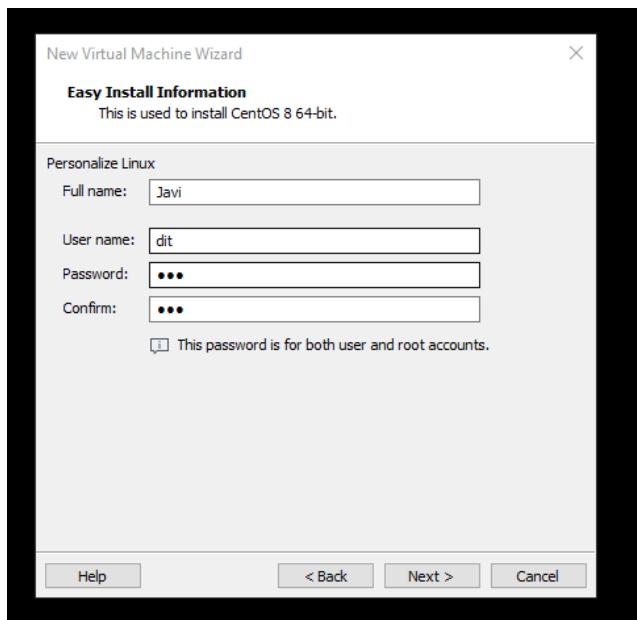


Figura A.0.3 - Creación de usuario máquina virtual

- 4) Se le asigna un nombre a la máquina virtual y se especifica la ruta donde se guardarán sus archivos (Figura A.0.4).

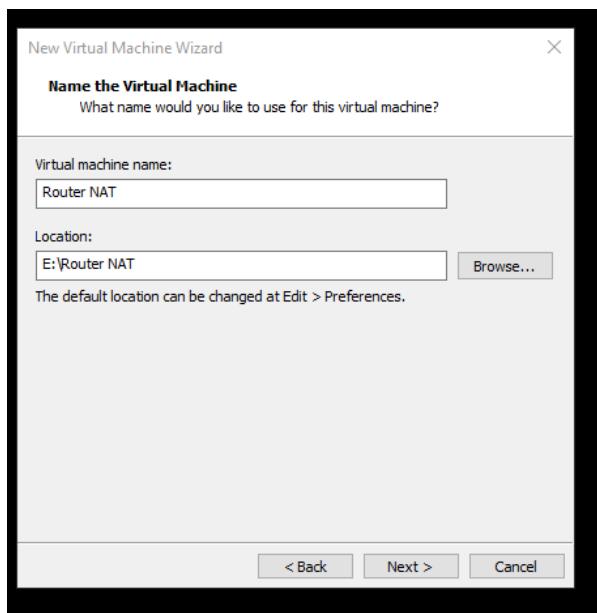


Figura A.0.4 - Nombre de la máquina y ruta de los archivos

- 5) A continuación, se especifica el tamaño de disco que se le desea dar a la máquina virtual y se usa la opción (*Split virtual disk in multiple files*) (Figura A.0.5).

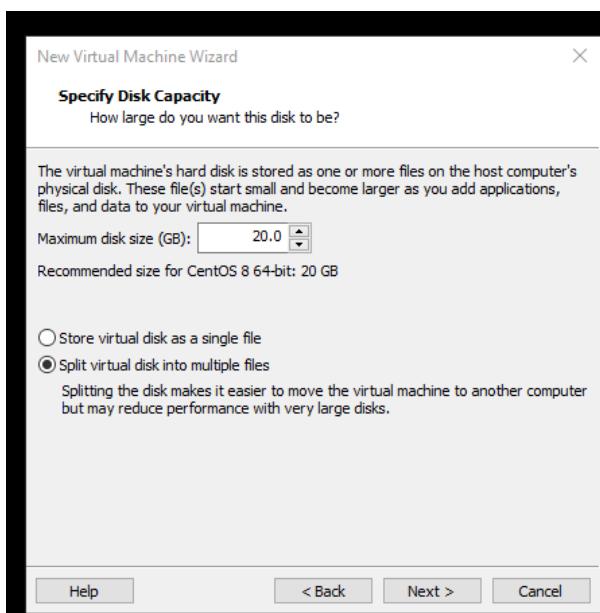


Figura A.0.5 - Espacio del disco máquina virtual

- 6) A continuación, se accede a personalizar las opciones de hardware (Figura A.0.6)

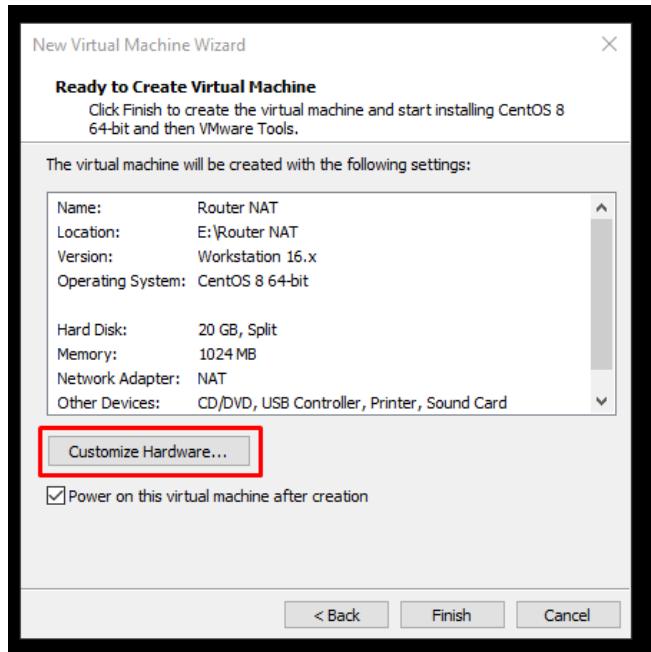


Figura A.0.6 - Personalizar hardware máquina virtual

- 7) Para crear las redes en las que se conectarán las máquinas virtuales se usará la opción *LAN segment*, en la cual se creará un segmento LAN y se le asignará a una nueva interfaz de red (Figura A.0.7).

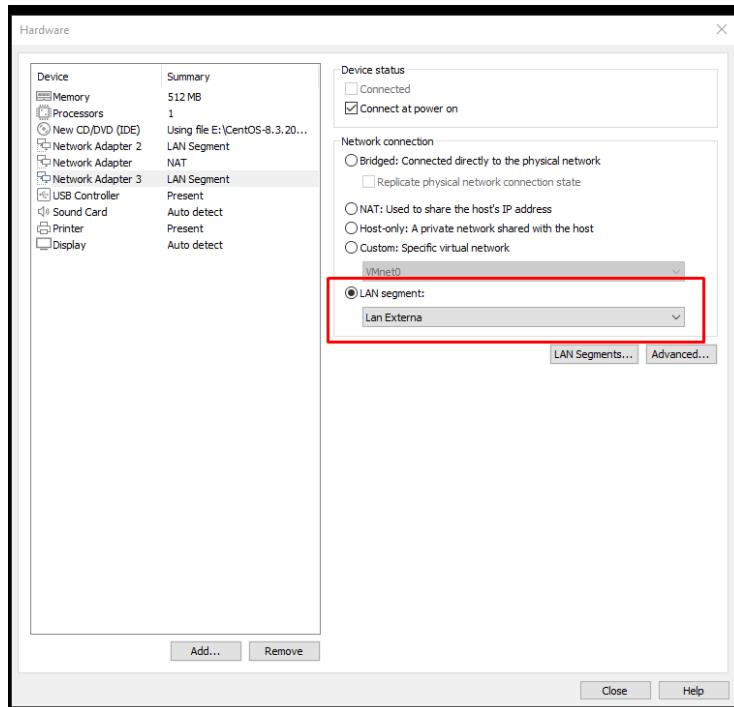


Figura A.0.7 - Añadir *LAN segment* a la máquina virtual

- 8) Una vez creada, se selecciona *Close* y en la nueva ventana se selecciona *Finish* para que comience la creación de la máquina virtual (Figura A.0.8).

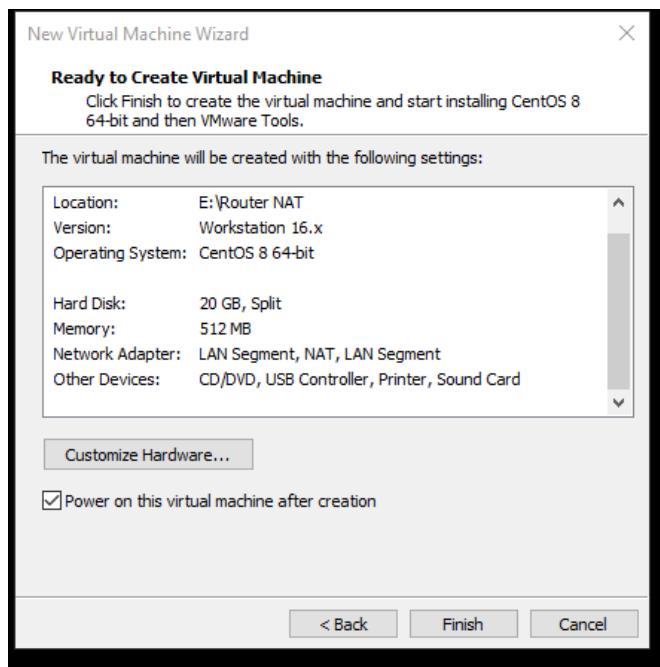


Figura A.0.8 - Fin de la creación de la máquina virtual

Wireshark y TCPdump

Estas dos herramientas son las que se han utilizado para poder capturar el tráfico de red generado.

En el equipo “Encaminador” se ha usado Wireshark, mientras que en los equipos con distribución CentOS se ha usado TCPdump.

Para instalarlas se usan los siguientes comandos:

```
Distribuciones: CentOS , Usuario: root

$ yum install tcpdump
$ yum install wireshark

Distribuciones: Debian , Usuario: root

$ apt update
$ apt upgrade -y
$ apt-get install tcpdump
$ apt-get install wireshark
```

En las Figura A.0.9 y Figura A.0.10, se observa cómo se puede realizar una captura en una determinada interfaz, usando Wireshark y TCPdump respectivamente.

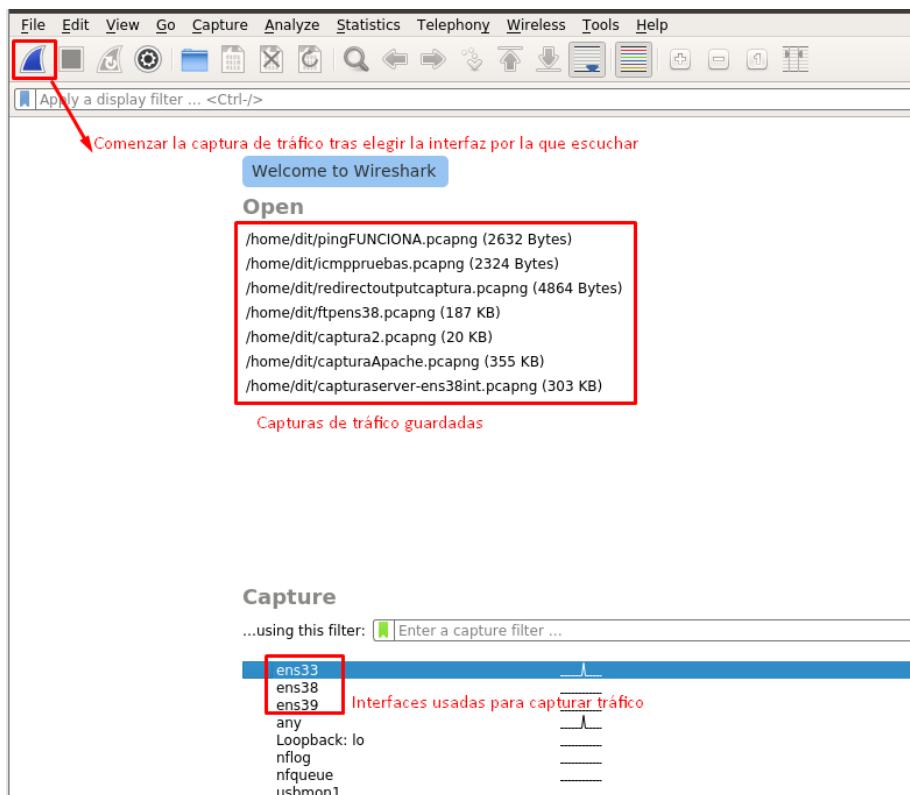


Figura A.0.9 - Interfaz Wireshark

```
root@localhost:/home/dit# tcpcdump -i ens33 -f -vvv
root@localhost:/home/dit# tcpcdump: listening on ens33, link-type EN10MB (Ethernet), capture size 262144 bytes
14:56:14.620812 IP (tos 0x0, ttl 64, id 32270, offset 0, flags [DF], proto ICMP (1), length 84) Paquete capturado
    localhost > 8.8.8.8: ICMP echo request, id 2170, seq 1, length 64
14:56:14.635940 IP (tos 0x0, ttl 64, id 17268, offset 0, flags [DF], proto UDP (17), length 74)
    localhost.59988 > 8.8.8.8.domain: [bad udp cksm 0x3b89 -> 0x3ee4!] 42729+ PTR? 137.106.168.192.in-addr.arpa. (46)
14:56:14.653936 ARP, Ethernet (len 6), IPv4 (len 4), Request who-has localhost tell gateway, length 46
14:56:14.653956 ARP, Ethernet (len 6), IPv4 (len 4), Reply localhost is-at 00:0c:29:f0:01:14 (oui Unknown), length 28
14:56:14.654099 IP (tos 0x0, ttl 128, id 63373, offset 0, flags [none], proto UDP (17), length 74)
    8.8.8.8.domain > localhost.59988: [udp sum ok] 42729 NXDomain q: PTR? 137.106.168.192.in-addr.arpa. 0/0/0 (46)
14:56:14.657094 IP (tos 0x0, ttl 64, id 17271, offset 0, flags [DF], proto UDP (17), length 72)
    localhost.38082 > 8.8.8.8.domain: [bad udp cksm 0x3b87 -> 0x0a5e!] 26172+ PTR? 2.106.168.192.in-addr.arpa. (44)
14:56:14.657597 IP (tos 0x0, ttl 128, id 63374, offset 0, flags [none], proto UDP (17), length 72)
    8.8.8.8.domain > localhost.38082: [udp sum ok] 26172 NXDomain q: PTR? 2.106.168.192.in-addr.arpa. 0/0/0 (44)
14:56:14.688855 IP (tos 0x0, ttl 128, id 63375, offset 0, flags [none], proto ICMP (1), length 84)
    8.8.8.8 > localhost: ICMP echo reply, id 2170, seq 1, length 64
```

Figura A.0.10 - Interfaz TCPdump

Hping3, Curl

Desde los sistemas CentOS se han instalado dos herramientas para generar tráfico de red, la primera de ellas es Hping3, la cual puede generar cualquier tipo de tráfico de red y la segunda de ellas es Curl, que nos permite realizar peticiones TCP a una dirección dada.

Para instalarlas se han usado los siguientes comandos.

Equipo: PC1/PC2	Usuario: dit
<pre>\$ su root \$ yum install hping3 \$ yum install curl</pre>	

A continuación, se detallarán los comandos más usados a la hora de realizar las pruebas en los distintos escenarios.

Equipo: PC1/PC2 **Usuario:**root

```
$ curl Ip_Encaminador
# Para realizar la petición de una página web
$ hping3 -c 1 --icmp-type echo-reply IP
# Para lanzar un paquete de tipo echo-reply a la dirección IP dada
$ hping3 -c 1 --icmp-type X --icmp-code Y IP
# Lanzar un paquete ICMP cuyo tipo es determinado por X y su código
determinado por Y a la dirección IP
$ hping3 -c 1 -S -A IP -p
# Lanzar un paquete TCP SYN-ACK a la dirección IP y al puerto P
$ hping3 -c 1 -S 10.10.1.10 -p 80
# Lanzar un paquete TCP SYN a la dirección 10.10.1.10 y al puerto 80
$ hping3 -c 1 -F IP -p
# Lanzar un paquete TCP FIN a la dirección IP y al puerto P
$ hping3 -c 1 -S --tcp-timestamp 10.10.1.10 -p 80
# Lanzar un paquete TCP SYN mostrando su timestamp a la dirección
#10.10.1.10 y al puerto 80
```


ANEXO B - INSTALACIÓN NFTABLES

En este anexo detallaremos como se realiza la instalación, usando el código fuente de la herramienta nftables en la distribución Debian.

Lo primero que tenemos que hacer antes de proceder a la instalación, es la actualización de la lista de paquetes.

Equipo:Encaminador **Usuario:**dit

```
$ sudo su  
$ apt-get update  
$ apt-upgrade
```

Instalación de dependencias

Una vez actualizada la lista de paquetes, comenzaremos a instalar las dependencias necesarias para instalar nftables.

La siguiente instalación, va a realizarse en una máquina con una distribución Debian Strech lanzada en 2017.

1. [Equipo:Encaminador - Usuario:root] Instalación de Libnftnl.

Equipo:Encaminador **Usuario:**root

```
$ mkdir CarpetaNFTables  
$ cd CarpetaNFTables  
$ git clone git://git.netfilter.org/libnftnl  
$ cd libnftnl  
$ sh autogen.sh  
$ ./configure  
$ make  
$ make install  
$ cd ..
```

2. [Equipo:Encaminador - Usuario:root] Instalación de Libnml.

Equipo:Encaminador **Usuario:**root

```
$ git clone git://git.netfilter.org/libnml  
$ cd libnml  
$ sh autogen.sh  
$ ./configure  
$ make  
$ make install  
$ cd ..
```

3. [Equipo:Encaminador - Usuario:root] Instalación de Bison.

```
Equipo:Encaminador Usuario:root
```

```
$ git clone https://git.savannah.gnu.org/git/bison.git
$ cd bison
$ sh autogen.sh
$ ./configure
$ make
$ make install
$ cd ..
```

4. [Equipo:Encaminador - Usuario:root] Instalación de Flex.

```
Equipo:Encaminador Usuario:root
```

```
$ wget http://prdownloads.sourceforge.net/flex/flex-2.5.33.tar.gz? /download
$ tar -xvzf flex-2.5.33.tar.gz
$ cd flex-2.5.33
$ sh autogen.sh
$ ./configure --prefix=/usr/local/flex
$ make
$ make install
$ cd ..
```

5. [Equipo:Encaminador - Usuario:root] Instalación de Asciidoc.

```
Equipo:Encaminador Usuario:root
```

```
$ git clone https://github.com/asciidoc/asciidoc-py3 asciidoc-9.0.4
$ cd asciidoc-9.0.4
$ sh autogen.sh
$ ./configure
$ make
$ make install
$ cd ..
```

6. [Equipo:Encaminador - Usuario:root] Instalación de Libgmp.

```
Equipo:Encaminador Usuario:root
```

```
$ wget https://gmplib.org/download/gmp/gmp-6.1.2.tar.xz
$ tar xvf gmp-6.1.2.tar.xz
$ cd gmp-6.1.2
$ ./configure --prefix=/usr/local/gmp/6_1_2
$ make
$ make install
```

7. [Equipo:Encaminador - Usuario:root] Instalación de Libreadline.

```
Equipo:Encaminador Usuario:root
```

```
$ wget ftp://ftp.gnu.org/gnu/readline/readline-8.0.tar.gz
$ tar xvfz readline-8.0.tar.gz
$ cd readline-8.0
$ ./configure --prefix=/usr/local/readline/8_0
$ make
$ make install
```

8. [Equipo:Encaminador - Usuario:root] Comprobación de dependencias.

Una vez hemos llegado a este punto, comprobamos que tenemos instaladas las dependencias necesarias para poder instalar el paquete de nftables sin que salte ningún aviso ni error.

Para ello comprobamos con las siguientes órdenes si los paquetes se encuentran instalados (Figura B-0.1 y Figura B.0.2)

```
Equipo:Encaminador Usuario:root
```

```
$ dpkg -l | grep libmn
$ dpkg -l | grep libnft
$ dpkg -l | grep bison
$ dpkg -l | grep flex
```

```
dit@linux:~$ dpkg -l |grep libmn
ii  libmn0:amd64          1.0.4-2                     amd64      minimalistic Netlink communication library
dit@linux:~$ dpkg -l |grep libnft
ii  libnftnl7:amd64       1.0.9-2                     amd64      Netfilter nftables userspace API library
dit@linux:~$ dpkg -l |grep bison
ii  bison                 2:3.0.4.dfsg-1build1   amd64      YACC-compatible parser generator
ii  libbison-dev:amd64    2:3.0.4.dfsg-1build1   amd64      YACC-compatible parser generator - development library
dit@linux:~$ dpkg -l |grep flex
ii  flex                  2.6.4-6                     amd64      fast lexical analyzer generator
ii  libfl-dev:amd64        2.6.4-6                     amd64      static library for flex (a fast lexical analyzer generator)
ii  libfl2:amd64           2.6.4-6                     amd64      SHARED library for flex (a fast lexical analyzer generator)
dit@linux:~$
```

Figura B-0.1 - Librerías nft(1)

```
Equipo:Encaminador Usuario:root
```

```
$ dpkg -l | grep readline
$ dpkg -l | grep libgm
```

```
dit@linux:~$ dpkg -l |grep readline
ii  libreadline7:amd64     7.0-3                     amd64      GNU readline and history libraries, run-time libraries
ii  readline-common        7.0-3                     all       GNU readline and history libraries, common files
dit@linux:~$ dpkg -l |grep libgm
ii  libgimme-3.0-0:amd64   3.2.0-1                   amd64      MIME message parser and creator library
ii  libomp10:amd64         2:6.1.2+dfsg-2            amd64      Multiprecision arithmetic library
dit@linux:~$
```

Figura B.0.2 - Librerías nft(2)

Instalación de Nftables

La mayoría de las anteriores dependencias, son librerías que ayudan a la interpretación de las reglas y al manejo de las mismas, ya que el núcleo de Netfilter viene por defecto instalado en las distribuciones.

A continuación, pasamos a descargar el framework de nftables.

1. [Equipo:Encaminador - Usuario:root] Instalación de Nftables.

Equipo:Encaminador **Usuario:**root

```
$ git clone git://git.netfilter.org/nftables
$ cd nftables
$ sh autogen.sh
$ ./configure
$ make
$ make install
```

2. Comprobación de instalación

Comprobamos que el paquete ha sido instalado, para ello hacemos uso del siguiente comando (Figura B.0.3).

Equipo:Encaminador **Usuario:**root

```
$ dpkg -l | grep nftables
```

```
dit@linux:~$ dpkg -l |grep nftables
ii  iptables-nftables-compat
ii  libnftnl7:amd64
ii  nftables
dit@linux:~$
```

Figura B.0.3 - nft instalado

3. [Equipo:Encaminador - Usuario:root] Cargar módulo nftables.

Una vez lo hayamos comprobado, cargamos su módulo

Equipo:Encaminador **Usuario:**root

```
$ modprobe nf_tables
```

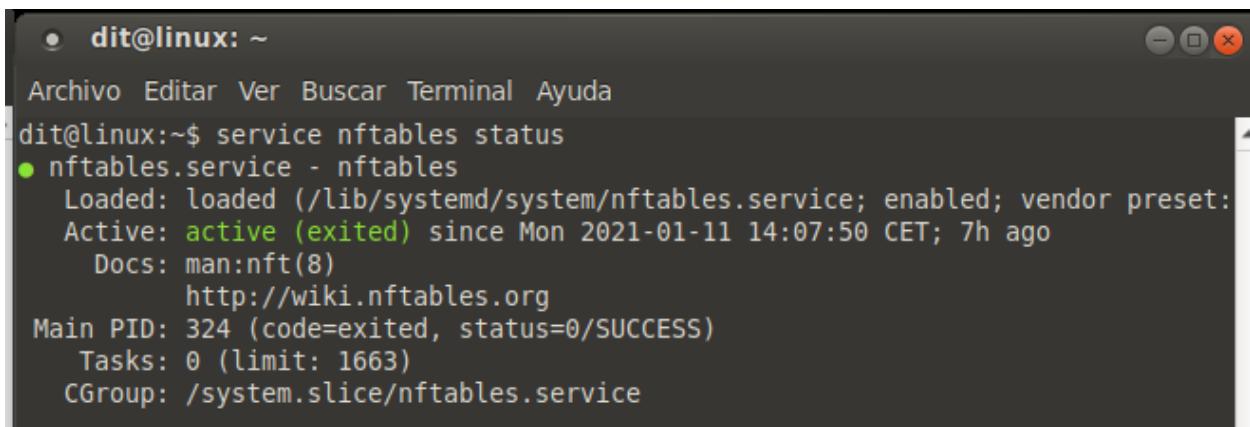
Con esto, ya tendríamos instalada nuestra herramienta en la distribución.

4. [Equipo:Encaminador - Usuario:root] Revisar estado del servicio.

Para comprobarlo revisamos el estado del servicio, introduciendo el siguiente comando y comprobando que el resultado es similar al mostrado en la Figura B.0.4.

Equipo:Encaminador **Usuario:**dit

```
$ service nftables status
```



dit@linux: ~

Archivo Editar Ver Buscar Terminal Ayuda

```
dit@linux:~$ service nftables status
● nftables.service - nftables
  Loaded: loaded (/lib/systemd/system/nftables.service; enabled; vendor preset:
  Active: active (exited) since Mon 2021-01-11 14:07:50 CET; 7h ago
    Docs: man:nft(8)
          http://wiki.nftables.org
   Main PID: 324 (code=exited, status=0/SUCCESS)
     Tasks: 0 (limit: 1663)
    CGroup: /system.slice/nftables.service
```

Figura B.0.4 - Servicio nft

ANEXO C - INSTALACIÓN CONNTRACK-TOOLS

En este anexo se explica el proceso de instalación de Conntrack-tools.

Lo primero que tenemos que realizar es la actualización de la lista de paquetes.

Equipo:Encaminador **Usuario:**dit

```
$ sudo su  
$ apt-get update  
$ apt-upgrade
```

Una vez realizado, instalamos las dependencias de Conntrack-tools.

Instalación dependencias de Conntrack-tools

1. [Equipo:Encaminador - Usuario:root] Instalación de Libnetlink.

Equipo:Encaminador **Usuario:**root

```
$ wget http://www.netfilter.org/projects/libnfnetlink/files \  
/libnfnetlink-0.0.11.tar.bz2  
$ tar xvf libnfnetlink-0.0.11.tar.bz2  
$ cd libnfnetlink-0.0.11  
$ ./configure  
$ make  
$ make install
```

2. [Equipo:Encaminador - Usuario:root] Instalación de Libnetfilter_conntrack.

Equipo:Encaminador **Usuario:**root

```
$ wget http://www.netfilter.org/projects/ libnetfilter_conntrack/ \  
files/libnetfilter_conntrack-1.0.8.tar.bz2  
$ tar xvf libnetfilter_conntrack-1.0.8.tar.bz2  
$ cd libnetfilter_conntrack-1.0.8  
$ ./configure  
$ make  
$ make install
```

3. [Equipo:Encaminador - Usuario:root] Comprobar módulos.

Para comprobar que están instalados los módulos necesarios, usamos el siguiente comando y comprobamos que su resultado es el mostrado en la Figura C.0.1.

Equipo:Encaminador **Usuario:**root

```
$ lsmod | grep conn
```

```
● root@localhost: /home/dit
Archivo Editar Ver Buscar Terminal Ayuda
root@localhost:/home/dit# lsmod | grep conn
nf_conntrack_netlink    40960  0
nf_conntrack           135168  1 nf_conntrack_netlink
libcrc32c              16384  1 nf_conntrack
nfnetlink              16384  2 nf_conntrack_netlink,nf_tables
root@localhost:/home/dit#
```

Figura C.0.1 - Módulos Conntrack-tools

Instalación de Conntrack-tools

Por último, instalamos Conntrack-tools.

1. [Equipo:Encaminador - Usuario:root] Instalación de Conntrack-tools.

Equipo: Encaminador **Usuario:** root

```
$ wget http://www.netfilter.org/projects/conntrack-tools/ \
files/conntrack-tools-1.4.6.tar.bz2
$ tar xvjf conntrack-tools-1.4.6.tar.bz2
$ cd conntrack-tools-1.4.6
$ ./configure
$ make
$ make install
```

2. [Equipo:Encaminador - Usuario:root] Comprobación de la instalación.

Para comprobar que este se ha instalado correctamente, ejecutamos el siguiente comando con la opción -L, para que nos muestre la tabla de flujos de conexión, que deberá estar vacía (Figura C.0.2).

Equipo: Encaminador **Usuario:** root

```
$ conntrack -L
```

```
● root@localhost: /home/dit
Archivo Editar Ver Buscar Terminal Ayuda
root@localhost:/home/dit# conntrack -L
conntrack v1.4.6 (conntrack-tools): 0 flow entries have been shown.
root@localhost:/home/dit#
```

Figura C.0.2 - Conntrack-tools instalado

ANEXO D - INSTALACIÓN Y CONFIGURACIÓN FIREWALLD

En este anexo, veremos cómo se realiza la instalación de Firewalld mediante el código fuente.

La primera dependencia con la que nos encontramos es desktop-file-utils.

Este paquete nos permite trabajar con una base de datos tipo MIME, usando como entrada la línea de comandos. Dado que las zonas de Firewalld son guardadas en estas bases, necesitamos este paquete para no obtener error a la hora de intentar configurarlas desde la línea de comandos.

Repositorio:

Equipo: Encaminador **Usuario:** dit

```
$ sudo apt-get update  
$ sudo apt-get install desktop-file-utils
```

Instalación de dependencias Firewalld

1. [Equipo:Encaminador - Usuario:root] Instalación de Desktop-File-Utils.

Equipo: Encaminador **Usuario:** root

```
$ mkdir firewall-dependencias  
$ cd firewall-dependencias  
$ wget ftp://www.freedesktop.org/software/desktop- \  
file-utils/releases/desktop-file-utils-0.26.tar.xz.sha256sum  
$ tar xvzf file-utils-0.26.tar.xz  
$ cd file-utils-0.26  
$ ./configure  
$ make  
$ make install  
$ cd ..
```

2. [Equipo:Encaminador - Usuario:root] Instalación de Gettext.

Este paquete es el encargado de ofrecernos las herramientas de programación que necesita nuestro frontend (Firewalld).

1) Ggettext por repositorio:

Equipo: Encaminador **Usuario:** dit

```
$ sudo apt-get update  
$ sudo apt-get install gettext
```

2) Gettext por código fuente:

Equipo: Encaminador **Usuario:** root

```
$ wget      ftp      https://ftp.gnu.org/pub/gnu/gettext/gettext-0.21.tar.gz
$ tar xvfz gettext-0.21.tar.gz
$ cd gettext-0.21
$ ./configure
$ make
$ make install
$ cd ..
```

3. [Equipo:Encaminador - Usuario:root] Instalación de Intltool.

La siguiente dependencia a instalar consta de varios sub-paquetes, es por ello que elegimos instalarla mediante el uso de repositorios, para evitar la complejidad del código fuente.

Repositorio:

Equipo: Encaminador **Usuario:** dit

```
$ sudo apt-get update
$ sudo apt-get install intltool
```

Se deben instalar las siguientes opciones de cmake-extras:

- *Cmake*
- *Cmake-data*
- *Libcurl4*
- *Libjsonpp1*
- *Librhash0*
- *Libuv1*

4. [Equipo:Encaminador - Usuario:root] Instalación de Glib.

La siguiente dependencia se llama glib, y en caso de instalarla por repositorio deberemos indicar su nombre tal y como se indica a continuación, mientras que, si se instala desde código fuente por git , bastará con usar los comandos indicados.

1) Glib2 por repositorio.

Equipo: Encaminador **Usuario:** dit

```
$ sudo apt-get update
$ sudo apt-get install libntrack-glib2
```

2) Glib2 por código fuente.

Equipo: Encaminador **Usuario:** root

```
$ git clone https://gitlab.gnome.org/GNOME/glib  
$ cd glib  
$ ./configure  
$ make  
$ make install  
$ cd ..
```

Existen 3 dependencias más, pero entendemos que si se usa la distribución indicada ya vienen de base, aun así, se indica a continuación por si en algún caso no estuviesen.

- *Docbook*
- *Libxslt*
- *Ebtables*
- *Ipsec*

El motivo de instalar ipsec y ebtables es que la funcionalidad de firewalld todavía no es completa, por lo que alguno de sus módulos todavía trabaja con iptables.

Una vez seguido los pasos anteriores procederemos a instalar firewalld.

Instalación de Firewalld

1. [Equipo:Encaminador - Usuario:root] Instalación de Firewalld.

Equipo: Encaminador **Usuario:** root

```
$ wget ftp://github.com/firewalld/firewalld/releases/ \  
download/v0.9.3/firewalld-0.9.3.tar.gz  
$ tar xvfz firewalld-0.9.3.tar.gz  
$ cd firewalld-0.9.3  
$ ./configure  
$ make  
$ make install  
$ cd ..
```

2. [Equipo:Encaminador - Usuario:root] Comprobar servicio Firewalld.

A continuación, para comprobar que el funcionamiento es correcto, que está cargado y que lo tenemos corriendo en nuestro equipo, introducimos el siguiente comando.

Comprobaremos que la salida producida se corresponde con las vistas en la Figura D.0.1 y Figura D.0.2.

Equipo: Encaminador **Usuario:** dit

```
#Estado del servicio  
$ systemctl start firewalld  
$ systemctl enable firewalld
```

```
$ service firewalld status
#Estado del Firewall
$ firewall-cmd --state
```

```
File Edit View Search Terminal Help
[root@localhost dit]# service firewalld status
Redirecting to /bin/systemctl status firewalld.service
● firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2021-01-12 03:17:07 PST; 28min ago
     Docs: man:firewalld(1)
     Main PID: 1022 (firewalld)
        Tasks: 2 (limit: 4761)
       Memory: 9.1M
      CGroup: /system.slice/firewalld.service
              └─1022 /usr/libexec/platform-python -s /usr/sbin/firewalld --nofork --nopid

Jan 12 03:17:05 localhost.localdomain systemd[1]: Starting firewalld - dynamic firewall daemon...
Jan 12 03:17:07 localhost.localdomain systemd[1]: Started firewalld - dynamic firewall daemon.
Jan 12 03:17:08 localhost.localdomain firewalld[1022]: WARNING: AllowZoneDrifting is enabled. This is considered >
```

Figura D.0.1 - Servicio firewalld

```
[root@localhost dit]# firewall-cmd --state
running
[root@localhost dit]#
```

Figura D.0.2 - Estado Firewalld

Configuración de Firewalld

Una vez llegados a este punto se procede a la configuración inicial de firewalld.

1. **[Equipo:Encaminador - Usuario:root]** Se establece la zona por defecto y se asocian las interfaces a dicha zona, en este caso se selecciona la zona “internal”.

Equipo:Encaminador **Usuario:**root

```
$ firewall-cmd --set-default-zone=internal
$ firewall-cmd --zone=internal --change-interface=ens37
$ firewall-cmd --zone=home --change-interface=ens33
```

2. **[Equipo:Encaminador - Usuario:root]** Se añaden los servicios de forma permanente a la zona “internal”.

Equipo:Encaminador **Usuario:**root

```
$      firewall-cmd      --permanent      --zone=intrenal      --add-
source=10.10.1.0/24
$ firewall-cmd --zone=internal --add-service=http --permanent
$ firewall-cmd --zone=internal --add-service=https --permanent
$ firewall-cmd --zone=internal --add-service=tls --permanent
$ firewall-cmd --zone=internal --add-service=dns --permanent
$ firewall-cmd --zone=internal --add-service=dhcp --permanent
$ firewall-cmd --zone=internal --add-service=ssh --permanent
```

3. [Equipo:Encaminador - Usuario:root] Se añaden los puertos a la zona “internal”.

```
Equipo:Encaminador Usuario:root
```

```
$ firewall-cmd --zone=internal --add-port=80/tcp --permanent
$ firewall-cmd --zone=internal --add-port=8080/tcp --permanent
$ firewall-cmd --zone=internal --add-port=22/tcp --permanent
```

4. [Equipo:Encaminador - Usuario:root] Se añaden los servicios de forma permanente a la zona “home”.

```
Equipo:Encaminador Usuario:root
```

```
$ firewall-cmd --zone=home --add-service=http --permanent
$ firewall-cmd --zone= home --add-service=https --permanent
$ firewall-cmd --zone=home --add-service=tls --permanent
$ firewall-cmd --zone=home --add-service=dns --permanent
$ firewall-cmd --zone=home --add-service=dhcp --permanent
$ firewall-cmd --zone=home --add-service=radius --permanent
```

5. [Equipo:Encaminador - Usuario:root] Se añaden los puertos a la zona “home”.

```
Equipo:Encaminador Usuario:root
```

```
$ firewall-cmd --zone=home --add-port=80/tcp --permanent
$ firewall-cmd --zone=home --add-port=8080/tcp --permanent
$ firewall-cmd --zone=home --add-port=22/tcp --permanent
$ firewall-cmd --zone=internal --add-rich-rule 'rule family="ipv4"
source      address=10.10.1.10 accept'
$ firewall-cmd --zone=home --add-port=53/tcp --permanent
$ firewall-cmd --zone=home --add-port=1812/tcp --permanent
$ firewall-cmd --zone=home --add-port=21/udp --permanent
$ firewall-cmd --zone=home --add-port=23/udp --permanent
```

Obteniendo modificadas así las siguientes partes del archivo de configuración.

Firewalld.conf

```
table inet firewalld {
    ct helper helper-netbios-ns-udp {
        type "netbios-ns" protocol udp
        l3proto ip
    }

    chain raw_PREROUTING {
        type filter hook prerouting priority raw + 10; policy accept;
        icmpv6 type { nd-router-advert, nd-neighbor-solicit } accept
        meta nfproto ipv6 fib saddr . iif oif missing drop
        jump raw_PREROUTING_ZONES_SOURCE
        jump raw_PREROUTING_ZONES
    }

    chain raw_PREROUTING_ZONES_SOURCE {
    }
```

```

chain raw_PREROUTING_ZONES {
    iifname "ens33" goto raw_PRE_internal
    iifname "ens37" goto raw_PRE_internal
    goto raw_PRE_internal
}

chain mangle_PREROUTING {
    type filter hook prerouting priority mangle + 10; policy accept;
    jump mangle_PREROUTING_ZONES_SOURCE
    jump mangle_PREROUTING_ZONES
}

chain mangle_PREROUTING_ZONES_SOURCE {
}

chain mangle_PREROUTING_ZONES {
    iifname "ens33" goto mangle_PRE_internal
    iifname "ens37" goto mangle_PRE_internal
    goto mangle_PRE_internal
}

chain filter_INPUT {
    type filter hook input priority filter + 10; policy accept;
    ct state { established, related } accept
    ct status dnat accept
    iifname "lo" accept
    jump filter_INPUT_ZONES_SOURCE
    jump filter_INPUT_ZONES
    ct state { invalid } drop
    reject with icmpx type admin-prohibited
}

chain filter_FORWARD {
    type filter hook forward priority filter + 10; policy accept;
    ct state { established, related } accept
    ct status dnat accept
    iifname "lo" accept
    ip6 daddr { ::/96, ::ffff:0.0.0.0/96, 2002::/24,
2002:a00::/24, 2002:7f00::/24, 2002:a9fe::/32, 2002:ac10::/28,
2002:c0a8::/32, 2002:e000::/19 } reject with icmpv6 type addr-unreachable
    jump filter_FORWARD_IN_ZONES_SOURCE
    jump filter_FORWARD_IN_ZONES
    jump filter_FORWARD_OUT_ZONES_SOURCE
    jump filter_FORWARD_OUT_ZONES
    ct state { invalid } drop
    reject with icmpx type admin-prohibited
}

chain filter_OUTPUT {
    type filter hook output priority filter + 10; policy accept;
    oifname "lo" accept
    ip6 daddr { ::/96, ::ffff:0.0.0.0/96, 2002::/24,
2002:a00::/24, 2002:7f00::/24, 2002:a9fe::/32, 2002:ac10::/28,
2002:c0a8::/32, 2002:e000::/19 } reject with icmpv6 type addr-unreachable
}

chain filter_INPUT_ZONES_SOURCE {
}

chain filter_INPUT_ZONES {

```

```

        iifname "ens33" goto filter_IN_home
        iifname "ens37" goto filter_IN_internal
        goto filter_IN_internal
    }

    chain filter_FORWARD_IN_ZONES_SOURCE {
    }

    chain filter_FORWARD_IN_ZONES {
        iifname "ens33" goto filter_FWDI_home
        iifname "ens37" goto filter_FWDI_internal
        goto filter_FWDI_internal
    }

    chain filter_FORWARD_OUT_ZONES_SOURCE {
    }

    chain filter_FORWARD_OUT_ZONES {
        oifname "ens33" goto filter_FWDO_home
        oifname "ens37" goto filter_FWDO_internal
        goto filter_FWDO_internal
    }

    chain filter_IN_internal {
        jump filter_IN_internal_pre
        jump filter_IN_internal_log
        jump filter_IN_internal_deny
        jump filter_IN_internal_allow
        jump filter_IN_internal_post
        meta l4proto { icmp, ipv6-icmp } accept
    }
        chain filter_IN_home {
        jump filter_IN_home_pre
        jump filter_IN_home_log
        jump filter_IN_home_deny
        jump filter_IN_home_allow
        jump filter_IN_home_post
        meta l4proto { icmp, ipv6-icmp } accept
    }

    chain filter_IN_internal_allow {
        tcp dport 22 ct state { new, untracked } accept
        ip daddr 224.0.0.251 udp dport 5353 ct state { new, untracked }
    } accept
        ip6 daddr ff02::fb udp dport 5353 ct state { new, untracked }
    accept
        udp dport 137 ct helper set "helper-netbios-ns-udp"
        udp dport 137 ct state { new, untracked } accept
        udp dport 138 ct state { new, untracked } accept
        ip6 daddr fe80::/64 udp dport 546 ct state { new, untracked }
    accept
        tcp dport 9090 ct state { new, untracked } accept
        tcp dport 80 ct state { new, untracked } accept
        tcp dport 443 ct state { new, untracked } accept
    }
    chain filter_IN_home_allow {
        tcp dport 21 ct helper set "helper-ftp-standard"
        tcp dport 21 ct state { new, untracked } accept
        tcp dport 22 ct state { new, untracked } accept
        tcp dport 23 ct helper set "helper-ftp-standard"
    }

```

```

        ip daddr 224.0.0.251 udp dport 5353 ct state { new, untracked }
} accept
ip6 daddr ff02::fb udp dport 5353 ct state { new, untracked }
accept
    udp dport 137 ct helper set "helper-netbios-ns-udp"
    udp dport 137 ct state { new, untracked } accept
    udp dport 138 ct state { new, untracked } accept
    ip6 daddr fe80::/64 udp dport 546 ct state { new, untracked }
accept
    tcp dport 9090 ct state { new, untracked } accept
    tcp dport 80 ct state { new, untracked } accept
    tcp dport 8080 ct state { new, untracked } accept
    tcp dport 443 ct state { new, untracked } accept
    udp dport 53 ct state { new, untracked } accept
    udp dport 1812 ct state { new, untracked } accept
}

chain filter_IN_internal_post {
}

chain filter_FWDI_internal {
    jump filter_FWDI_internal_pre
    jump filter_FWDI_internal_log
    jump filter_FWDI_internal_deny
    jump filter_FWDI_internal_allow
    jump filter_FWDI_internal_post
    meta l4proto { icmp, ipv6-icmp } accept
}

chain mangle_PRE_internal {
    jump mangle_PRE_internal_pre
    jump mangle_PRE_internal_log
    jump mangle_PRE_internal_deny
    jump mangle_PRE_internal_allow
    jump mangle_PRE_internal_post
}

}

table ip firewalld {
    chain nat_PREROUTING {
        type nat hook prerouting priority dstnat + 10; policy accept;
        jump nat_PREROUTING_ZONES_SOURCE
        jump nat_PREROUTING_ZONES
    }

    chain nat_PREROUTING_ZONES_SOURCE {
    }

    chain nat_PREROUTING_ZONES {
        iifname "ens33" goto nat_PRE_internal
        iifname "ens37" goto nat_PRE_internal
        goto nat_PRE_internal
    }

    chain nat_POSTROUTING {
        type nat hook postrouting priority srcnat + 10; policy accept;
        jump nat_POSTROUTING_ZONES_SOURCE
        jump nat_POSTROUTING_ZONES
    }

    chain nat_POSTROUTING_ZONES_SOURCE {
    }
}

```

```

chain nat_POSTROUTING_ZONES {
    oifname "ens33" goto nat_POST_internal
    oifname "ens37" goto nat_POST_internal
    goto nat_POST_internal
}

chain nat_PRE_internal {
    jump nat_PRE_internal_pre
    jump nat_PRE_internal_log
    jump nat_PRE_internal_deny
    jump nat_PRE_internal_allow
    jump nat_PRE_internal_post
}

chain nat_POST_internal {
    jump nat_POST_internal_pre
    jump nat_POST_internal_log
    jump nat_POST_internal_deny
    jump nat_POST_internal_allow
    jump nat_POST_internal_post
}

}

table ip6 firewalld {
    chain nat_PREROUTING {
        type nat hook prerouting priority dstnat + 10; policy accept;
        jump nat_PREROUTING_ZONES_SOURCE
        jump nat_PREROUTING_ZONES
    }

    chain nat_PREROUTING_ZONES_SOURCE {
    }

    chain nat_PREROUTING_ZONES {
        iifname "ens33" goto nat_PRE_internal
        iifname "ens37" goto nat_PRE_internal
        goto nat_PRE_internal
    }

    chain nat_POSTROUTING {
        type nat hook postrouting priority srcnat + 10; policy accept;
        jump nat_POSTROUTING_ZONES_SOURCE
        jump nat_POSTROUTING_ZONES
    }

    chain nat_POSTROUTING_ZONES_SOURCE {
    }

    chain nat_POSTROUTING_ZONES {
        oifname "ens33" goto nat_POST_internal
        oifname "ens37" goto nat_POST_internal
        goto nat_POST_internal
    }

    chain nat_PRE_internal {
        jump nat_PRE_internal_pre
        jump nat_PRE_internal_log
        jump nat_PRE_internal_deny
        jump nat_PRE_internal_allow
        jump nat_PRE_internal_post
    }
}

```

```
}

chain nat_POST_internal {
    jump nat_POST_internal_pre
    jump nat_POST_internal_log
    jump nat_POST_internal_deny
    jump nat_POST_internal_allow
    jump nat_POST_internal_post
}
}
```

ANEXO E - ESCENARIO 2 EQUIPOS

Escenario 2 Equipos

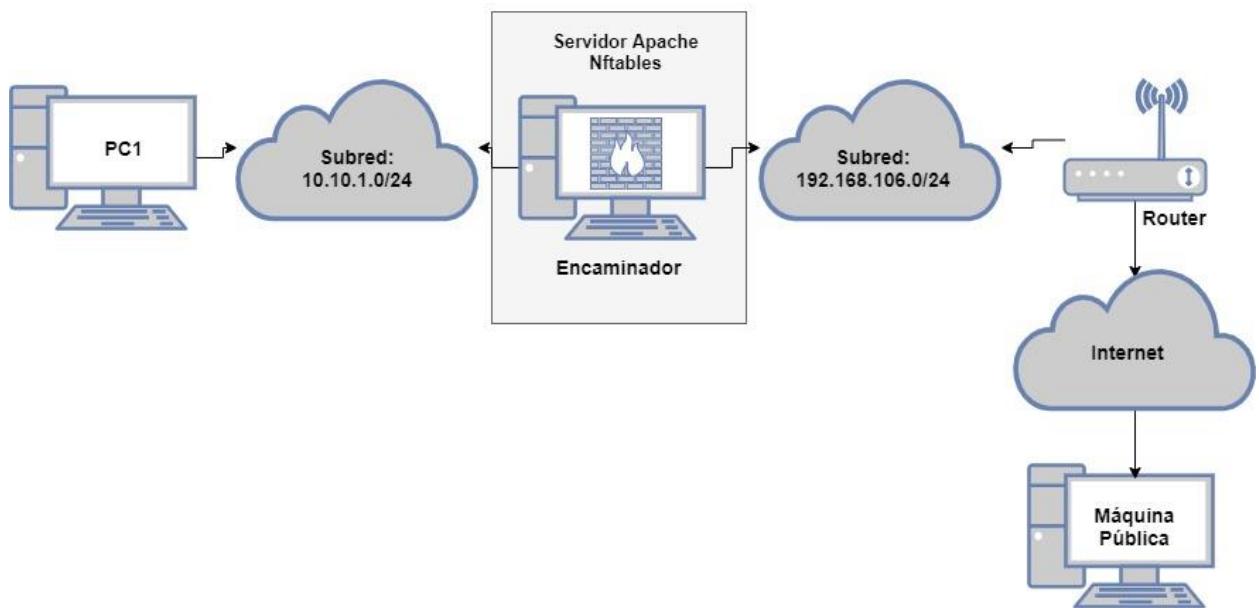


Figura E.0.1 - Esquema escenario 2 equipos

Este es el escenario que se plantea para usar dos equipos (Figura E.0.1).

Consta de un primer equipo, que será el cliente (PC1) y es el encargado de realizar las peticiones y generar tráfico de red y segundo equipo servidor (Encaminador) en el cual tendremos las herramientas de firewall para tratar el tráfico.

En la Tabla 19, se recoge un cuadro resumen, con los sistemas operativos, las interfaces y las direcciones Ip usadas en este escenario.

Notación	S.O usado	Interfaz	Ip/Mask
PC1	CentOS 8	Ens37	10.10.1.20/24
Encaminador	Debian- Stretch	Ens38 Ens33	10.10.1.10/24 192.168.106.128/24

Tabla 19 - Resumen escenario 2 equipos

A continuación, se detallarán tanto los comandos para configurar el entorno, como los archivos de configuración listos para sustituirlos en los equipos.

Todos los comandos que se van a indicar están ejecutados con el superusuário *root*.

Configuración del equipo PC1

Comandos configuración PC1

Equipo: PC1 **Usuario:** root

```
$ ip address flush dev ens37
$ ip link set ens37 down
$ ip address add 10.10.1.20/24 dev ens37
$ ip link set ens37 up
$ ip neigh flush all
$ ip address ls dev ens37
$ service network restart
```

La configuración puede realizarse con los comandos indicados o bien modificando el archivo *ifcfg* correspondiente en la ruta */etc/sysconfig/network-scripts/* y dejando su contenido de la siguiente forma.

```
[root@localhost dit]# cat /etc/sysconfig/network-scripts/ifcfg-ens37-1
HWADDR=00:0C:29:62:ED:AD
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
IPADDR=10.10.1.20
PREFIX=24
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV4_DNS_PRIORITY=100
IPV6INIT=yes
IPV6_AUTOCONF=no
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
IPV6_DNS_PRIORITY=100
NAME=ens37
UUID=98f4a976-7f4d-4708-9505-6cb0ee7024a5
DEVICE=ens37
ONBOOT=yes
```

Tras realizar las configuraciones indicadas pueden observarse las siguientes salidas.

- La Figura E.0.2 muestra el contenido del archivo *ifcfg* de PC1

Equipo: PC1 **Usuario:** root

```
$ cat /etc/sysconfig/network-scripts/ifcfg-ens37-1
```

- La Figura E.0.3 muestra la salida del comando *ip address*.

```
Equipo:PC1 Usuario:root
$ ip addr
```

- La Figura E.0.4 muestra la salida del comando *ip route*.

```
Equipo:PC1 Usuario:root
$ ip route
```

```
[root@localhost dit]# cat /etc/sysconfig/network-scripts/ifcfg-ens37-1
HWADDR=00:0C:29:62:ED:AD
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
IPADDR=10.10.1.20
PREFIX=24
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV4_DNS_PRIORITY=100
IPV6INIT=yes
IPV6_AUTOCONF=no
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
IPV6_DNS_PRIORITY=100
NAME=ens37
UUID=98f4a976-7f4d-4708-9505-6cb0ee7024a5
DEVICE=ens37
ONBOOT=yes
```

Figura E.0.2 - Archivo ifcfg de PC1 (escenario 2 equipos)

```
[root@localhost dit]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:62:ed:a3 brd ff:ff:ff:ff:ff:ff
        inet 192.168.106.133/24 brd 192.168.106.255 scope global dynamic noprefixroute ens33
            valid_lft 1498sec preferred_lft 1498sec
        inet6 fe80::174d:4044:89b9:ffd0/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
3: ens37: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:62:ed:ad brd ff:ff:ff:ff:ff:ff
        inet 10.10.1.20/24 brd 10.10.1.255 scope global noprefixroute ens37
            valid_lft forever preferred_lft forever
        inet6 fe80::28db:9db3:3f28:14d0/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
4: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
    link/ether 52:54:00:f0:3f:a8 brd ff:ff:ff:ff:ff:ff
        inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
            valid_lft forever preferred_lft forever
5: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr0 state DOWN group default qlen 1000
    link/ether 52:54:00:f0:3f:a8 brd ff:ff:ff:ff:ff:ff
```

Figura E.0.3 - Ip address PC1 (escenario 2 equipos)

```
[root@localhost dit]# ip route
default via 192.168.106.2 dev ens33 proto dhcp metric 100
10.10.1.0/24 dev ens37 proto kernel scope link src 10.10.1.20 metric 101
192.168.106.0/24 dev ens33 proto kernel scope link src 192.168.106.133 metric 100
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1 linkdown
[root@localhost dit]#
```

Figura E.0.4 - Ip route PC1 (escenario 2 equipos)

Configuración del equipo Encaminador

Pasamos a continuación a realizar la configuración del equipo Encaminador.

Comandos de configuración Encaminador

Equipo:Encaminador **Usuario:**root

```
$ ip address flush dev ens38
$ ip link set ens38 down
$ ip address add 10.10.1.10/24 dev ens38
$ ip link set ens38 up
$ ip neigh flush all
$ ip address ls dev ens38
$ service network restart
```

Arrancamos también los siguientes servicios.

Equipo:Encaminador **Usuario:**root

```
$ service nftables start
$ service firewalld stop
$ service httpd start
```

La configuración puede realizarse con los comandos indicados o bien modificando el archivo *interfaces* correspondiente en la ruta */etc/network/* y dejando su contenido de la siguiente forma.

```
root@linux:/home/dit# cat /etc/network/interfaces

# interfaces(5) file used by ifup(8) and ifdown(8)
#auto lo
#iface lo inet loopback
#
iface ens38 inet static
    address 10.10.1.10
    netmask 255.255.255.0
    network 10.10.1.0
    broadcast 10.10.1.255
    dns-nameservers 8.8.8.8 8.8.4.4
```

Si usamos este segundo método, una vez cargado el archivo tenemos que usar los siguientes comandos para que la interfaz se recargue y tome su nueva configuración.

Equipo: Encaminador **Usuario:** root

```
$ ifdown ens38  
$ ifup ens38
```

Se puede comprobar la configuración, observando las salidas de las siguientes figuras.

- La Figura E.0.5 muestra el contenido del archivo *interfaces* de Encaminador

Equipo: Encaminador **Usuario:** root

```
$ cat /etc/network/interfaces
```

- La Figura E.0.6 muestra la salida del comando *ip address*.

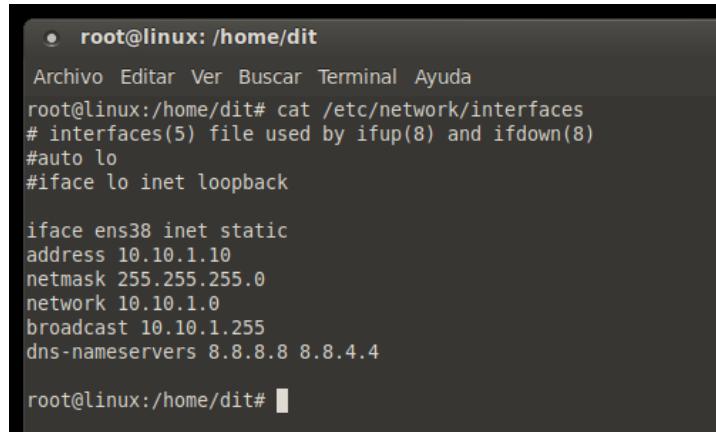
Equipo: Encaminador **Usuario:** root

```
$ ip addr
```

- La Figura E.0.7 muestra la salida del comando *ip route*.

Equipo: Encaminador **Usuario:** root

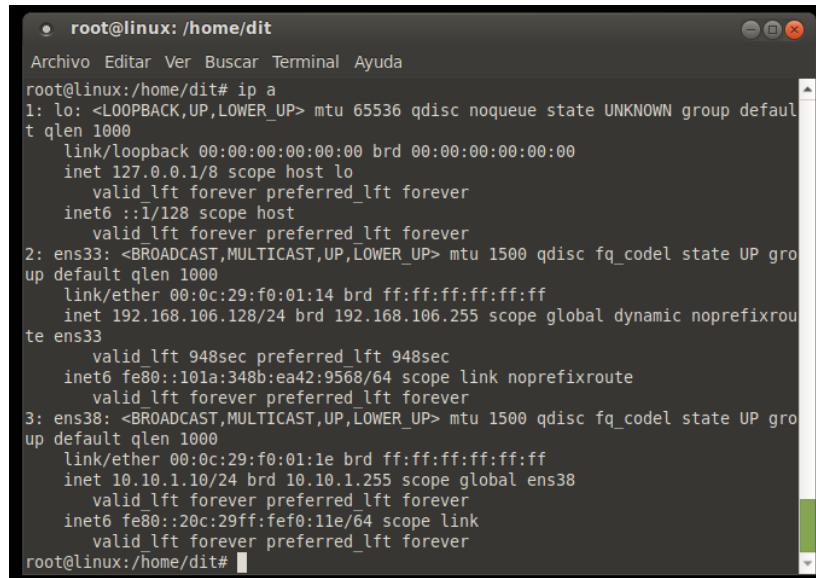
```
$ ip route
```



The screenshot shows a terminal window with a dark background and white text. The title bar says "root@linux: /home/dit". The window contains the following text:

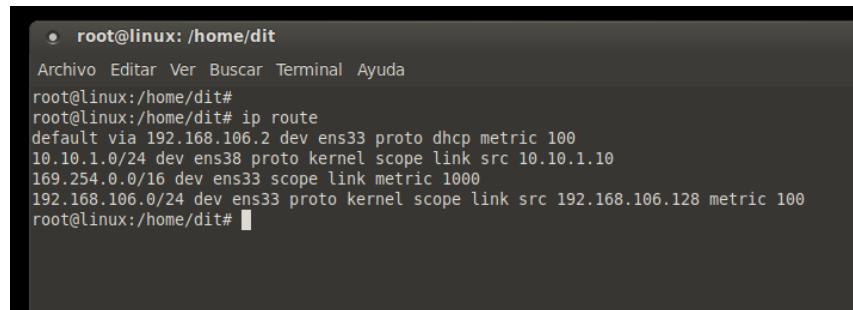
```
Archivo Editar Ver Buscar Terminal Ayuda  
root@linux:/home/dit# cat /etc/network/interfaces  
# interfaces(5) file used by ifup(8) and ifdown(8)  
#auto lo  
#iface lo inet loopback  
  
iface ens38 inet static  
address 10.10.1.10  
netmask 255.255.255.0  
network 10.10.1.0  
broadcast 10.10.1.255  
dns-nameservers 8.8.8.8 8.8.4.4  
  
root@linux:/home/dit#
```

Figura E.0.5 - Archivo interfaces Encaminador (escenario 2 equipos)



root@linux:/home/dit# ip a
Archivo Editar Ver Buscar Terminal Ayuda
root@linux:/home/dit# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
 inet 127.0.0.1/8 scope host lo
 valid_lft forever preferred_lft forever
 inet6 ::1/128 scope host
 valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
 link/ether 00:0c:29:f0:01:14 brd ff:ff:ff:ff:ff:ff
 inet 192.168.106.128/24 brd 192.168.106.255 scope global dynamic noprefixroute ens33
 valid_lft 948sec preferred_lft 948sec
 inet6 fe80::101a:348b:ea42:9568/64 scope link noprefixroute
 valid_lft forever preferred_lft forever
3: ens38: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
 link/ether 00:0c:29:f0:01:1e brd ff:ff:ff:ff:ff:ff
 inet 10.10.1.10/24 brd 10.10.1.255 scope global ens38
 valid_lft forever preferred_lft forever
 inet6 fe80::20c:29ff:fe0:11e/64 scope link
 valid_lft forever preferred_lft forever
root@linux:/home/dit#

Figura E.0.6 - Ip address Encaminador (escenario 2 equipos)



root@linux:/home/dit#
Archivo Editar Ver Buscar Terminal Ayuda
root@linux:/home/dit# ip route
root@linux:/home/dit# ip route
default via 192.168.106.2 dev ens33 proto dhcp metric 100
10.10.1.0/24 dev ens38 proto kernel scope link src 10.10.1.10
169.254.0.0/16 dev ens33 scope link metric 1000
192.168.106.0/24 dev ens33 proto kernel scope link src 192.168.106.128 metric 100
root@linux:/home/dit#

Figura E.0.7 - Ip route (escenario 2 equipos)

ANEXO F - ESCENARIO 3 EQUIPOS

Escenario 3 Equipos

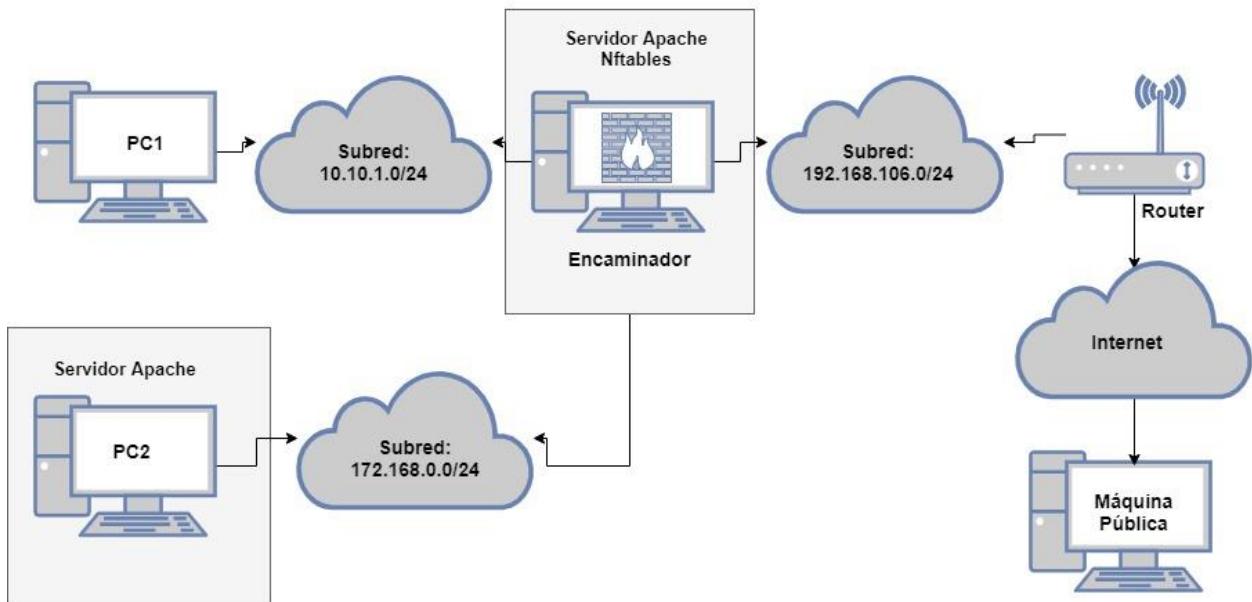


Figura F.0.1 - Esquema escenario 3 equipos

Este es el escenario que se plantea para usar tres equipos (Figura F.0.1).

Consta de un primer equipo que será el Pc de la red privada (PC1) y que tiene un servidor Apache escuchando en su puerto 80. Un equipo servidor (Encaminador) en el cual tendremos las herramientas de firewall para tratar el tráfico, así como otro servidor Apache en escucha en el puerto 80 y por último tenemos un Pc cliente (PC2), el cual está en la red pública, y que será el encargado de generar tráfico de red.

En la Tabla 20, se recoge un cuadro resumen, con los sistemas operativos, las interfaces y las direcciones Ip usadas en este escenario.

Notación	S.O usado	Interfaz	Ip/Mask
PC1	CentOS 8	Ens37	10.10.1.20/24
PC2	CentOS 8	Ens37	172.168.0.20/24
Encaminador	Debian- Stretch	Ens38 Ens33 Ens39	10.10.1.10/24 192.168.106.128/24 172.168.0.10/24

Tabla 20 - Resumen escenario 3 equipos

A continuación, se detallarán tanto los comandos para configurar el entorno, como los archivos de configuración listos para sustituirlos en los equipos.

Todos los comandos que se van a indicar están ejecutados con el superusuario *root*.

Configuración del equipo PC1

Comandos de configuración PC1:

```
Equipo:PC1 Usuario:root

$ ip address flush dev ens37
$ ip link set ens37 down
$ ip address add 10.10.1.20/24 dev ens37
$ ip link set ens37 up
$ ip neigh flush all
$ ip address ls dev ens37
$ service network restart
$ service httpd start
```

La configuración puede realizarse con los comandos indicados o bien modificando el archivo *ifcfg* correspondiente en la ruta */etc/sysconfig/network-scripts/* y dejando su contenido de la siguiente forma.

```
[root@localhost dit]# cat /etc/sysconfig/network-scripts/ifcfg-ens37-
1
HWADDR=00:0C:29:62:ED:AD
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
IPADDR=10.10.1.20
PREFIX=24
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV4_DNS_PRIORITY=100
IPV6INIT=yes
IPV6_AUTOCONF=no
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
IPV6_DNS_PRIORITY=100
NAME=ens37
UUID=98f4a976-7f4d-4708-9505-6cb0ee7024a5
DEVICE=ens37
ONBOOT=yes
```

Tras realizar las configuraciones indicadas pueden observarse las siguientes salidas.

- La Figura F.0.2 muestra el contenido del archivo *ifcfg* de PC1

```
Equipo:PC1 Usuario:root
$ cat /etc/sysconfig/network-scripts/ifcfg-ens37-1
```

- La Figura F.0.3 muestra la salida del comando *ip address*.

```
Equipo:PC1 Usuario:root
$ ip addr
```

- La Figura F.0.4 muestra la salida del comando *ip route*.

```
Equipo:PC1 Usuario:root
$ ip route
```

```
[root@localhost dit]# cat /etc/sysconfig/network-scripts/ifcfg-ens37-1
HWADDR=00:0C:29:62:ED:AD
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
IPADDR=10.10.1.20
PREFIX=24
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV4_DNS_PRIORITY=100
IPV6INIT=yes
IPV6_AUTOCONF=no
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
IPV6_DNS_PRIORITY=100
NAME=ens37
UUID=98f4a976-7f4d-4708-9505-6cb0ee7024a5
DEVICE=ens37
ONBOOT=yes
```

Figura F.0.2 - Archivo ifcfg de PC1 (escenario 3 equipos)

```
[root@localhost dit]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:62:ed:a3 brd ff:ff:ff:ff:ff:ff
        inet 192.168.106.133/24 brd 192.168.106.255 scope global dynamic noprefixroute ens33
            valid_lft 1498sec preferred_lft 1498sec
        inet6 fe80::174d:4044:89b9:ffd0/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
3: ens37: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:62:ed:ad brd ff:ff:ff:ff:ff:ff
        inet 10.10.1.20/24 brd 10.10.1.255 scope global noprefixroute ens37
            valid_lft forever preferred_lft forever
        inet6 fe80::28db:9db3:3f28:14d0/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
4: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
    link/ether 52:54:00:f0:3f:a8 brd ff:ff:ff:ff:ff:ff
        inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
            valid_lft forever preferred_lft forever
5: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr0 state DOWN group default qlen 1000
    link/ether 52:54:00:f0:3f:a8 brd ff:ff:ff:ff:ff:ff
```

Figura F.0.3 - Ip address PC1 (escenario 3 equipos)

```
[root@localhost dit]# ip route
default via 192.168.106.2 dev ens33 proto dhcp metric 100
10.10.1.0/24 dev ens37 proto kernel scope link src 10.10.1.20 metric 101
192.168.106.0/24 dev ens33 proto kernel scope link src 192.168.106.133 metric 100
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1 linkdown
[root@localhost dit]#
```

Figura F.0.4 - Ip route PC1 (escenario 3 equipos)

Configuración del equipo PC2

Tras haber configurado el equipo PC1, pasamos a configurar PC2

Comandos de configuración PC2

Equipo: PC2 Usuario: root
\$ ip address flush dev ens37 \$ ip link set ens37 down \$ ip address add 172.168.0.20/24 dev ens37 \$ ip link set ens37 up \$ ip neigh flush all \$ ip address ls dev ens37 \$ service network restart

La configuración puede realizarse con los comandos indicados o bien modificando el archivo *ifcfg* correspondiente en la ruta */etc/sysconfig/network-scripts/* y dejando su contenido de la siguiente forma.

[root@localhost dit]# cat /etc/sysconfig/network-scripts/ifcfg-ens37
HWADDR=00:0C:29:91:4B:F2 TYPE=Ethernet PROXY_METHOD=none BROWSER_ONLY=no BOOTPROTO=none

```
IPADDR=172.168.0.20
PREFIX=24
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV4_DNS_PRIORITY=100
IPV6INIT=yes
IPV6_AUTOCONF=no
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
IPV6_DNS_PRIORITY=100
NAME=ens37
UUID=98f4a976-7f4d-4708-9505-6cb0ee7024a5
DEVICE=ens37
ONBOOT=yes
```

Tras realizar las configuraciones indicadas pueden observarse las siguientes salidas.

- La Figura F.0.5 muestra el contenido del archivo *ifcfg* de PC1

Equipo: PC2 **Usuario:** root

```
$ cat /etc/sysconfig/network-scripts/ifcfg-ens37
```

- La Figura F.0.6 muestra la salida del comando *ip address*.

Equipo: PC2 **Usuario:** root

```
$ ip addr
```

- La Figura F.0.7 muestra la salida del comando *ip route*.

Equipo: PC2 **Usuario:** root

```
$ ip route
```

```
dit@localhost:/home/dit
File Edit View Search Terminal Help
[root@localhost dit]# cat /etc/sysconfig/network-scripts/ifcfg-ens37
HWADDR=00:0C:29:91:4B:F2
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
IPADDR=172.168.0.20
PREFIX=24
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV4_DNS_PRIORITY=100
IPV6INIT=no
NAME=ens37
UUID=9c3dacd4-2bcc-4fb6-94e0-aa8696f49e57
DEVICE=ens37
ONBOOT=yes
[root@localhost dit]#
```

Figura F.0.5 - Archivo ifcfg de PC2 (escenario 3 equipos)

```
dit@localhost:/home/dit
File Edit View Search Terminal Help
[root@localhost dit]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:91:4b:e8 brd ff:ff:ff:ff:ff:ff
        inet 192.168.106.135/24 brd 192.168.106.255 scope global dynamic noprefixroute ens33
            valid_lft 1533sec preferred_lft 1533sec
        inet6 fe80::6840:d633:108c:1680/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
3: ens37: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:91:4b:f2 brd ff:ff:ff:ff:ff:ff
        inet 172.168.0.20/24 scope global ens37
            valid_lft forever preferred_lft forever
        inet6 fe80::20c:29ff:fe91:4bf2/64 scope link
            valid_lft forever preferred_lft forever
4: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
    link/ether 52:54:00:2c:f0:dc brd ff:ff:ff:ff:ff:ff
        inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
            valid_lft forever preferred_lft forever
5: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr0 state DOWN group default qlen 1000
    link/ether 52:54:00:2c:f0:dc brd ff:ff:ff:ff:ff:ff
[root@localhost dit]#
```

Figura F.0.6 - Ip address PC2 (escenario 3 equipos)

```
dit@localhost:/home/dit
File Edit View Search Terminal Help
[root@localhost dit]# ip route
default via 192.168.106.2 dev ens33 proto dhcp metric 100
172.168.0.0/24 dev ens37 proto kernel scope link src 172.168.0.20 metric 101
192.168.106.0/24 dev ens33 proto kernel scope link src 192.168.106.135 metric 100
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1 linkdown
[root@localhost dit]#
```

Figura F.0.7 - Ip route PC2 (escenario 3 equipos)

Configuración del equipo Encaminador

Pasamos a continuación a realizar la configuración del equipo Encaminador.

Comandos de configuración Encaminador

Equipo:Encaminador **Usuario:**root

```
$ ip address flush dev ens38
$ ip link set ens38 down
$ ip address add 10.10.1.10/24 dev ens38
$ ip link set ens38 up
$ ip neigh flush all
$ ip address ls dev ens38

$ ip address flush dev ens39
$ ip link set ens39 down
$ ip address add 172.168.0.10/24 dev ens39
$ ip link set ens39 up
$ ip neigh flush all
$ ip address ls dev ens39

$ service network restart
```

Arrancamos también los siguientes servicios.

Equipo:Encaminador **Usuario:**root

```
$ service nftables start
$ service firewalld stop
$ service httpd start
```

La configuración puede realizarse con los comandos indicados o bien modificando el archivo *interfaces* correspondiente en la ruta */etc/network/* y dejando su contenido de la siguiente forma.

```
root@linux:/home/dit# cat /etc/network/interfaces

# interfaces(5) file used by ifup(8) and ifdown(8)
#auto lo
#iface lo inet loopback
#
auto ens38
iface ens38 inet static
address 10.10.1.10
netmask 255.255.255.0
network 10.10.1.0
broadcast 10.10.1.255
dns-nameservers 8.8.8.8 8.8.4.4

auto ens39
iface ens39 inet static
address 172.168.0.10
netmask 255.255.255.0
network 172.168.0.0
broadcast 172.168.0.255
```

```
dns-nameservers 8.8.8.8 8.8.4.4
```

Si usamos este segundo método, una vez cargado el archivo tenemos que usar los siguientes comandos para que la interfaz se recargue y tome su nueva configuración.

Equipo: Encaminador **Usuario:** root

```
$ ifdown ens38
$ ifup ens38
$ ifdown ens39
$ ifup ens39
```

- La Figura F.0.8 muestra el contenido del archivo *interfaces* de Encaminador

Equipo: Encaminador **Usuario:** root

```
$ cat /etc/network/interfaces
```

- La Figura F.0.9 muestra la salida del comando *ip address*.

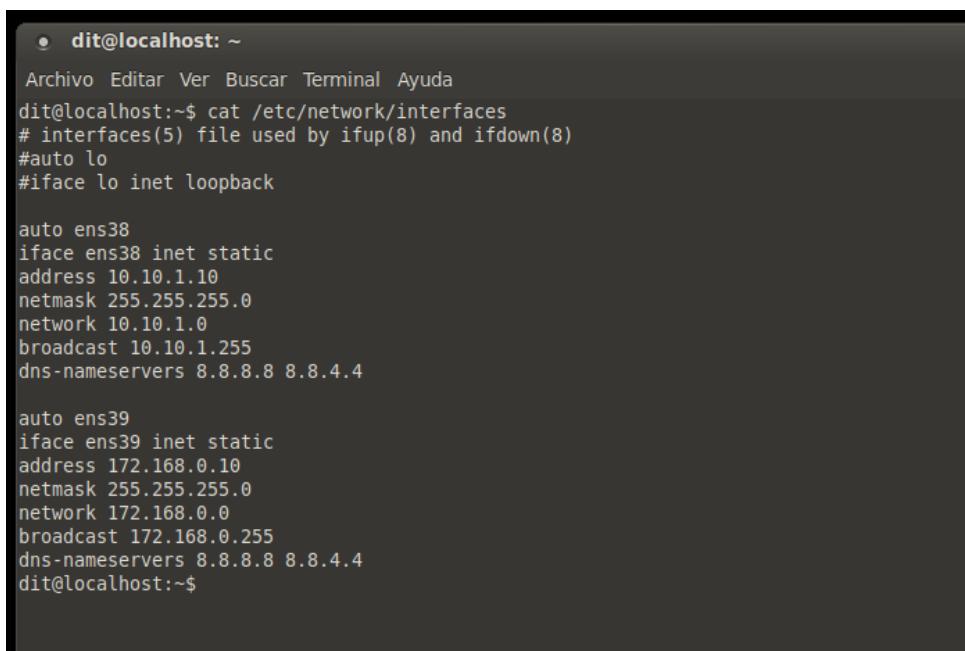
Equipo: Encaminador **Usuario:** root

```
$ ip addr
```

- La Figura F.0.10 muestra la salida del comando *ip route*.

Equipo: Encaminador **Usuario:** root

```
$ ip route
```



```
● dit@localhost: ~
Archivo Editar Ver Buscar Terminal Ayuda
dit@localhost:~$ cat /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)
#auto lo
#iface lo inet loopback

auto ens38
iface ens38 inet static
address 10.10.1.10
netmask 255.255.255.0
network 10.10.1.0
broadcast 10.10.1.255
dns-nameservers 8.8.8.8 8.8.4.4

auto ens39
iface ens39 inet static
address 172.168.0.10
netmask 255.255.255.0
network 172.168.0.0
broadcast 172.168.0.255
dns-nameservers 8.8.8.8 8.8.4.4
dit@localhost:~$
```

Figura F.0.8 - Archivo interfaces Encaminador (escenario 3 equipos)

```
• dit@localhost: ~
Archivo Editar Ver Buscar Terminal Ayuda
dit@localhost:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:f0:01:14 brd ff:ff:ff:ff:ff:ff
        inet 192.168.106.128/24 brd 192.168.106.255 scope global dynamic noprefixroute ens33
            valid_lft 1489sec preferred_lft 1489sec
        inet6 fe80::101a:348b:ea42:9568/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
3: ens38: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:f0:01:1e brd ff:ff:ff:ff:ff:ff
        inet 10.10.1.10/24 brd 10.10.1.255 scope global ens38
            valid_lft forever preferred_lft forever
        inet6 fe80::20c:29ff:fe0:11e/64 scope link
            valid_lft forever preferred_lft forever
4: ens39: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:f0:01:28 brd ff:ff:ff:ff:ff:ff
        inet 172.168.0.10/24 brd 172.168.0.255 scope global ens39
            valid_lft forever preferred_lft forever
        inet6 fe80::20c:29ff:fe0:128/64 scope link
            valid_lft forever preferred_lft forever
dit@localhost:~$
```

Figura F.0.9 - Ip address Encaminador (escenario 3 equipos)

```
• dit@localhost: ~
Archivo Editar Ver Buscar Terminal Ayuda
dit@localhost:~$ ip route
default via 192.168.106.2 dev ens33 proto dhcp metric 100
10.10.1.0/24 dev ens38 proto kernel scope link src 10.10.1.10
169.254.0.0/16 dev ens39 scope link metric 1000
172.168.0.0/24 dev ens39 proto kernel scope link src 172.168.0.10
192.168.106.0/24 dev ens33 proto kernel scope link src 192.168.106.128 metric 100
dit@localhost:~$
```

Figura F.0.10 - Ip route Encaminador (escenario 3 equipos)

Caso específico: Escenario NAT Asimétrico

El escenario de NAT asimétrico es un caso puntual en el que se usará el escenario de 3 equipos y se añadirá un cuarto equipo por el cual se enviará el tráfico de respuesta de PC2 (Figura F.0.11).

La configuración de este escenario como bien se ha dicho, parte de la base del escenario de 3 equipos - (Anexo F - Escenario 3 Equipos).

La configuración del cuarto equipo “Router NAT” es la siguiente:

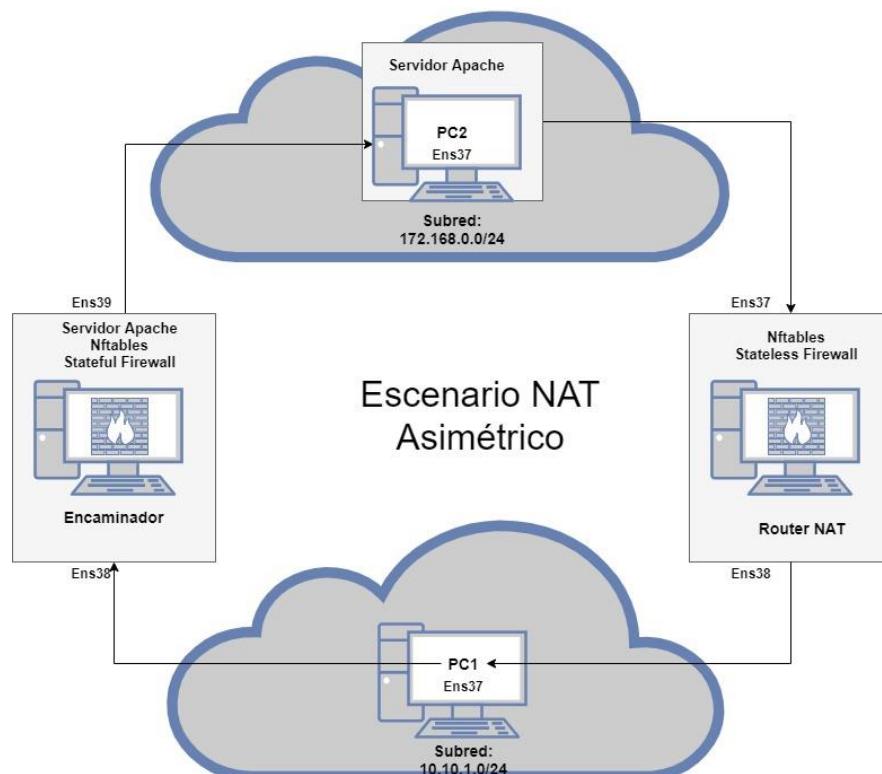


Figura F.0.11 - Esquema escenario NAT asimétrico

En la Tabla 21 se recoge un cuadro resumen, con los sistemas operativos, las interfaces y las direcciones Ip usadas en este escenario.

Notación	S.O usado	Interfaz	Ip/Mask
<i>PC1</i>	<i>CentOS 8</i>	<i>Ens37</i>	<i>10.10.1.20/24</i>
<i>PC2</i>	<i>CentOS 8</i>	<i>Ens37</i>	<i>172.168.0.20/24</i>
<i>Encaminador</i>	<i>Debian- Stretch</i>	<i>Ens38</i>	<i>10.10.1.10/24</i>
		<i>Ens33</i>	<i>192.168.106.128/24</i>
		<i>Ens39</i>	<i>172.168.0.10/24</i>
<i>Router NAT</i>	<i>CentOS 8</i>	<i>Ens37</i>	<i>10.10.1.30/24</i>
		<i>Ens38</i>	<i>172.168.0.40/24</i>

Tabla 21 - Resumen escenario NAT asimétrico

Configuración del equipo Router NAT

Comandos de configuración Router NAT

```
Equipo:Router NAT Usuario:root
```

```
$ ip address flush dev ens38
$ ip link set ens38 down
$ ip address add 10.10.1.30/24 dev ens37
$ ip link set ens38 up
$ ip neigh flush all
$ ip address ls dev ens38

$ ip address flush dev ens39
$ ip link set ens39 down
$ ip address add 172.168.0.400/24 dev ens38
$ ip link set ens39 up
$ ip neigh flush all
$ ip address ls dev ens39

$ service network restart
```

La configuración puede realizarse con los comandos indicados o bien modificando el archivo *ifcfg* correspondiente en la ruta */etc/sysconfig/network-scripts/* y dejando su contenido de la siguiente forma.

```
[root@localhost dit]# cat /etc/sysconfig/network-scripts/ifcfg-ens37

HWADDR=00:0C:29:08:2E:23
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
IPADDR=10.10.1.30
PREFIX=24
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=ens37
UUID=e07385a5-adcd-4c23-a426-812daa776be8
ONBOOT=yes
```

```
[root@localhost dit]# cat /etc/sysconfig/network-scripts/ifcfg-ens38

TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=none
```

```

IPADDR=172.168.0.40
PREFIX=24
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=ens38
UUID=861b6eb7-85a9-419e-ad9b-0e6aa4e2db15
ONBOOT=yes
HWADDR=00:0C:29:08:2E:2D

```

- La Figura F.0.12 muestra la salida del comando *ip address*.

Equipo: Encaminador **Usuario:** root
\$ ip addr

- La Figura F.0.13 muestra la salida del comando *ip route*.

Equipo: Encaminador **Usuario:** root
\$ ip rou

```

[root@localhost dit]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:08:2e:19 brd ff:ff:ff:ff:ff:ff
3: ens37: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:08:2e:23 brd ff:ff:ff:ff:ff:ff
    inet 10.10.1.30/24 brd 10.10.1.255 scope global noprefixroute ens37
        valid_lft forever preferred_lft forever
    inet6 fe80::5f1c:bbbc:2b47:b267/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
4: ens38: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:08:2e:2d brd ff:ff:ff:ff:ff:ff
    inet 172.168.0.40/24 brd 172.168.0.255 scope global noprefixroute ens38
        valid_lft forever preferred_lft forever
    inet6 fe80::b70:8368:212c:71b1/64 scope link noprefixroute
        valid_lft forever preferred_lft forever

```

Figura F.0.12 - Ip address (Router NAT)

```

[root@localhost dit]# ip rou
10.10.1.0/24 dev ens37 proto kernel scope link src 10.10.1.30 metric 101
172.168.0.0/24 dev ens38 proto kernel scope link src 172.168.0.40 metric 102
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1 linkdown
[root@localhost dit]#

```

Figura F.0.13 - Ip route (Router NAT)

ANEXO G - CHEAT SHEET NFTABLES

Línea de comandos CheatSheet

Comando	Opciones	Descripción/Ejemplo
table	<i>add</i>	Añadir una tabla nueva nft add table inet tabla_pru
	<i>delete</i>	Borrar una tabla nft delete table inet tabla_pru
	<i>flush</i>	Borrar el contenido de una tabla nft flush table inet tabla_pru
	<i>list</i>	Listar una tabla (se recomienda usar la opción -a o -n) -a: muestra el handle -n: muestra el número de la regla nft list table inet tabla_pru -a nft list table inet tabla_pru -n

Tabla 22 - CheatSheet comando table

Comando	Opciones	Descripción/Ejemplo
chain		Crear una cadena nueva
	<i>add</i>	nft add chain inet tabla_pru cadena_pru { type filter hook input priority 0 ; policy drop ; } nft add chain arp tabla_pru cadena_arp { type filter hook prerouting priority -300 ; policy accept ; }
	<i>create</i>	Solo crea la nueva cadena si esta no existe nft create chain inet tabla_pru cadena_new { type filter hook output priority 100 ; policy accept ; }
	<i>delete</i>	Borrar una cadena nft delete chain inet tabla_pru cadena_new

	<i>list</i>	Listar una cadena nft list chain inet tabla_pru cadena_new
	<i>flush</i>	Limpiar el contenido de una cadena nft flush chain inet tabla_pru cadena_new
	<i>rename</i>	Renombrar una cadena nft rename chain inet tabla_pru cadena_new cadena2

Tabla 23 - CheatSheet comando chain

Comando	Opciones	Descripción/Ejemplo
rule	<i>add</i>	Añadir una regla nueva tras la última existente nft add rule inet tabla_pru cadena_pru accept
	<i>insert</i>	Insertar una regla en una posición de la cadena determinada. Para ver las posiciones de las reglas se usa la opción -n al listar una cadena. nft insert rule inet tabla_pru cadena_pru <posición> accept
	<i>replace</i>	Reemplazar una regla existente por otra nueva. Para saber el número de handle cuando listemos una cadena se usa la opción -a nft replace rule inet tabla_pru cadena_pru <n_handle> accept
	<i>delete</i>	Borrar una regla nft delete rule inet tabla_pru cadena_pru <n_handle>

Tabla 24 - CheatSheet comando rule

Módulos CheatSheet

Módulo	Opciones	Descripción /Ejemplo
ip	<i>length</i>	Valor de longitud total del paquete nft add rule inet tabla_pru cadena_pru ip length 260 accept
	<i>id</i>	Ip ID del paquete nft add rule inet tabla_pru cadena_pru ip id !=20 accept
	<i>ttl</i>	TTL (time to live) del paquete nft add rule inet tabla_pru cadena_pru ip ttl { 74-128 } accept
	<i>protocol</i>	Protocolo de capa superior que lo encapsula nft add rule inet tabla_pru cadena_pru ip protocol { icmp, udp, tcp } accept
	<i>saddr</i>	Dirección origen nft add rule inet tabla_pru cadena_pru ip saddr 192.168.1.0/24
	<i>daddr</i>	Dirección destino nft add rule inet tabla_pru cadena_pru ip daddr 192.168.1.10 accept
	<i>version</i>	Versión del paquete IP nft add rule inet tabla_pru cadena_pru ip version 6 drop
	<i>checksum</i>	Checksum de la cabecera IP nft add rule inet tabla_pru cadena_pru ip checksum 1260 accept

Tabla 25 - CheatSheet módulo Ip

Módulo	Opciones	Descripción /Ejemplo
tcp	<i>dport</i>	Puerto destino del paquete TCP nft add rule inet tabla_pru cadena_pru tcp dport != 10-19 drop
	<i>sport</i>	Puerto origen del paquete TCP nft add rule inet tabla_pru cadena_pru tcp sport { 80, 8080 } accept
	<i>sequence</i>	Secuencia del paquete TCP nft add rule inet tabla_pru cadena_pru tcp sequence 1 accept
	<i>ackseq</i>	Secuencia del paquete TCP-ACK nft add rule inet tabla_pru cadena_pru tcp ackseq 1 accept
	<i>flags</i>	Flags del paquete TCP nft add rule inet tabla_pru cadena_pru tcp flags { fin, syn, ack } accept
	<i>checksum</i>	Checksum del paquete TCP nft add rule inet tabla_pru cadena_pru tcp checksum 1000 drop

Tabla 26 - CheatSheet módulo TCP

Módulo	Opciones	Descripción /Ejemplo
udp	<i>dport</i>	Puerto destino paquete UDP nft add rule inet tabla_pru cadena_pru udp dport 80 accept
	<i>sport</i>	Puerto origen paquete UDP nft add rule inet tabla_pru cadena_pru udp sport != 8080 accept
	<i>length</i>	Longitud total del paquete UDP nft add rule inet tabla_pru cadena_pru udp length 100-200 accept

	<i>checksum</i>	Checksum del paquete UDP nft add rule inet tabla_pru cadena_pru udp checksum 1280 drop
--	-----------------	---

Tabla 27 - CheatSheet módulo UDP

Módulo	Opciones	Descripción /Ejemplo
icmp	<i>type</i>	Tipo del paquete ICMP
		nft add rule inet tabla_pru cadena_pru icmp type { echo-reply, echo-request, router-solicitation } accept
	<i>code</i>	Código del paquete ICMP
		nft add rule inet tabla_pru cadena_pru icmp code != 4 accept
	<i>id</i>	Id del paquete ICMP
		nft add rule inet tabla_pru cadena_pru icmp code != 4 accept
	<i>sequence</i>	Secuencia del paquete ICMP
		nft add rule inet tabla_pru cadena_pru icmp sequence 1-4 accept
	<i>checksum</i>	Checksum del paquete ICMP
		nft add rule inet tabla_pru cadena_pru icmp checksum != { 1000, 999, 000 } accept
	<i>mtu</i>	(Maximun Transfer Unit) del paquete ICMP
		nft add rule inet tabla_pru cadena_pru icmp mtu 100-1264 accept
	<i>gateway</i>	Gateway de salida del paquete ICMP
		nft add rule inet tabla_pru cadena_pru icmp gateway 10.10.1.30 drop

Tabla 28 - CheatSheet módulo ICMP

Módulo	Opciones	Descripción /Ejemplo
ether	<i>saddr</i>	Mac de origen del paquete
		nft add rule inet tabla_pru cadena_pru ether saddr 00:5f:60:0c:ab:e4 accept
	<i>type</i>	Ethertype del paquete
		nft add rule inet tabla_pru cadena_pru ether type vlan drop

Tabla 29 - CheatSheet módulo ether

Módulo	Opciones	Descripción /Ejemplo
arp	<i>ptype</i>	Tipo del payload del paquete ARP
		nft add rule arp tabla_pru cadena_arp arp ptype 0x200 accept
	<i>htype</i>	Tipo de cabecera del paquete ARP
		nft add rule arp tabla_pru cadena_arp arp htype 1 drop
	<i>hlen</i>	Longitud de la cabecera ARP
		nft add rule arp tabla_pru cadena_arp arp hlen != { 0, 999, 664 } accept
	<i>plen</i>	Longitud del payload ARP
		nft add rule arp tabla_pru cadena_arp arp plen 100-500 accept
	<i>operation</i>	Tipo de operación del paquete ARP
		nft add rule arp tabla_pru cadena_arp arp operation { nak, inreplay, request, reply } accept

Tabla 30 - CheatSheet módulo ARP

Módulo	Opciones	Descripción /Ejemplo
ct	<i>state</i>	Estado que asigna el módulo ct
		nft add rule inet tabla_pru cadena_pru ct { new, established, related } accept
	<i>direction</i>	Dirección del paquete referente a la conexión(original/reply)
		nft add rule inet tabla_pru cadena_pru ct direction { original, reply } accept
	<i>status</i>	Estado interno de la conexión
		nft add rule inet tabla_pru cadena_pru ct status { reply, assured, confirmed } accept
	<i>mark</i>	Marca de la conexión
		nft add rule inet tabla_pru cadena_pru ct mark set 1 accept nft add rule inet tabla_pru cadena_pru ct mark != 1 drop
	<i>helper</i>	Helper para conexiones de 2 flujos
		nft add ct helper inet tabla_pru ftp { type "ftp" protocol tcp ; }
	<i>saddr/daddr</i>	Dirección origen y destino (original/reply)
		nft add rule inet tabla_pru cadena_pru ct original saddr 10.10.1.10 accept nft add rule inet tabla_pru cadena_pru ct reply saddr 204.160.24.50 drop
	<i>protocol</i>	Protocolo del paquete de la conexión (original/reply)
		nft add rule inet tabla_pru cadena_pru ct original protocol icmp drop nft add rule inet tabla_pru cadena_pru ct reply protocol tcp accept
	<i>proto-dts</i> <i>proto-src</i>	Puerto origen/destino del paquete de la conexión (original/reply)
		nft add rule inet tabla_pru cadena_pru ct original proto-dts 80 accept nft add rule inet tabla_pru cadena_pru ct reply proto-src 21 drop

Tabla 31 - CheatSheet módulo ct

Módulo	Opciones	Descripción /Ejemplo
meta	<i>iifname</i>	Nombre de la interfaz de entrada del paquete nft add rule inet tabla_pru cadena_pru meta iifname "ens33" drop
	<i>oifname</i>	Nombre de la interfaz de salida del paquete nft add rule inet tabla_pru cadena_pru meta oifname "lo" accept
	<i>iif</i>	Índice de la interfaz de entrada del paquete nft add rule inet tabla_pru cadena_pru meta iif ens38 accept
	<i>oif</i>	Índice de la interfaz de salida del paquete nft add rule inet tabla_pru cadena_pru meta oif ens39 accept
	<i>iiftype</i>	Tipo de interfaz de entrada nft add rule inet tabla_pru cadena_pru meta iiftype { loopback, ppp } drop
	<i>oiftype</i>	Tipo de interfaz de salida nft add rule inet tabla_pru cadena_pru meta oiftype { ipip, ipip6 } accept
	<i>length</i>	Tamaño del paquete nft add rule inet tabla_pru cadena_pru meta length > 2000 drop
	<i>protocol</i>	Protocolo Ethertype del paquete nft add rule inet tabla_pru cadena_pru meta protocol { ip, arp } accept
	<i>l4proto</i>	Protocolo de capa 4 del paquete nft add rule inet tabla_pru cadena_pru meta l4proto tcp accept
	<i>mark</i>	Marca del paquete nft add rule inet tabla_pru cadena_pru meta mark set 0x2 continue nft add rule inet tabla_pru cadena_pru meta mark 0x2 accept

	<i>pkttype</i>	Tipo del paquete
		nft add rule inet tabla_pru cadena_pru meta pkttype { broadcast, unicast, multicast } accept

Tabla 32 - CheatSheet módulo meta

Conntrack-Tools CheatSheet

Comando	Opciones	Descripción/Ejemplo
conntrack	-L	Imprime la lista con la tabla de conexiones conntrack -L
	-G	Busca en la tabla de conexiones por un determinado patrón dado conntrack -G -p tcp --dport 80
	-D	Borra una entrada de la tabla de conexiones conntrack -D -p tcp --dport 80
	-E	Muestra en tiempo real los logs de la tabla de conexiones conntrack -E
	-F	Borra toda la tabla de conexiones conntrack -F
	-C	Muestra el número de entradas de la tabla de conexiones conntrack -C

Tabla 33 - CheatSheet Conntrack-Tools

ANEXO H - REPASO PROTOCOLOS (TCP,UDP,ICMP)

En este anexo se realizará un repaso de los conceptos fundamentales de los protocolos TCP, UDP e ICMP, así como los tipos de mensajes de cada uno de estos, sus flags, sus cabeceras etc...

Comenzaremos repasando el protocolo TCP.

Protocolo TCP (Transmission Control Protocol)

TCP son las siglas de “Protocolo de Control de Transmisión”, el cual se encuentra definido en la RFC 793 y su unidad básica recibe el nombre de segmento.

Este protocolo opera en el nivel 4 del modelo TCP/IP, basándose para ello en el protocolo IP de capa 3.

La función principal de este protocolo es asegurar que los datos que son enviados, sean recibidos por el otro extremo en orden y sin errores y en caso de que exista algún error, encargarse de retransmitir el paquete.

Esto lo consigue mediante un establecimiento de conexión, es decir, antes de realizar el envío de datos, tiene lugar una fase de negociación en la que el emisor y el receptor anuncian y acuerdan las bases para la comunicación, así como abrir una sesión para poder comunicar sus datos.

Por esto, TCP es un protocolo orientado a conexión, debido a que antes de enviar los datos, establece una fase de negociación con el otro punto de comunicación.

A continuación, se mostrará las diferentes partes de una cabecera TCP.

Cabecera TCP

0	1	2	3		
0	1	2	3		
4	5	6	7		
8	9	10	11		
12	13	14	15		
16	17	18	19		
20	21	22	23		
24	25	26	27		
28	29	30	31		
Puerto TCP origen		Puerto TCP destino			
Número de Secuencia					
Número de Asentimiento(ACK)					
Offset	Reservado	URG	ACK		
		PSH	RST		
		SYN	FIN		
Ventana					

Checksum	Puntero Urgente
Opciones	

Tabla 34 - Cabecera TCP

En la Tabla 34, se puede observar la composición de una cabecera TCP.

Los primeros 20 bytes son conocidos como “cabecera fija”, a continuación, se dará una breve explicación de cada campo.

- **Puerto TCP origen:** Identificador del puerto emisor (16 bits)
- **Puerto TCP destino:** Identificador del puerto receptor (16 bits)
- **Número de Secuencia:** Identifica los segmentos dentro del flujo de conexión TCP (32 bits)
- **Número de Asentimiento:** Identifica el siguiente número de secuencia que el emisor espera recibir (32 bits)
- **Offset:** Identifica el tamaño de la cabecera (4 bits)
- **Reservado:** Reservado para un uso futuro, su valor debe estar a 0 (6 bits)
- **Flag URG:** Bit que activa el envío urgente (Prioriza el procesado de los segmentos marcados con este bit) (1bit)
- **Flag ACK:** Bit que indica que el paquete contiene un asentimiento (1 bit)
- **Flag PSH:** Bit que activa la función Push (El receptor procesará los segmentos conforme estos lleguen) (1bit)
- **Flag RST:** Bit que se envía cuando el receptor recibe un mensaje que no esperaba y que no debería haber recibido del emisor (1bit)
- **Flag SYN:** Bit que indica el comienzo de la sincronización en el establecimiento de la conexión (1 bit)
- **Flag FIN:** Bit que indica que no quedan más datos que transmitir en el origen (1 bit)
- **Ventana:** Idéntica el tamaño de la ventana de recepción (16 bits)
- **Checksum:** Suma de comprobación usada para verificar si existen errores (16 bits)
- **Puntero Urgente:** Identifica el número de bytes desde que se activa el flag URG hasta que este termina (16 bits)
- **Opciones:** Este campo añade opciones que la cabecera fija no es capaz de cubrir (tamaño variable, siendo este un múltiplo de 8 bits).

Una vez visto los diferentes campos que componen la cabecera TCP se procederá a ver las fases del establecimiento y cierre de conexión y los segmentos intercambiados durante cada fase.

Establecimiento de la conexión TCP

Esta fase inicial, se conoce con el nombre de *3-way-handshake* y es la fase de negociación antes de establecer la comunicación de datos entre dos entidades.

En este punto nos referiremos al Cliente como la entidad que inicia la conexión y como Servidor a aquella entidad con la que quiere comunicarse el cliente.

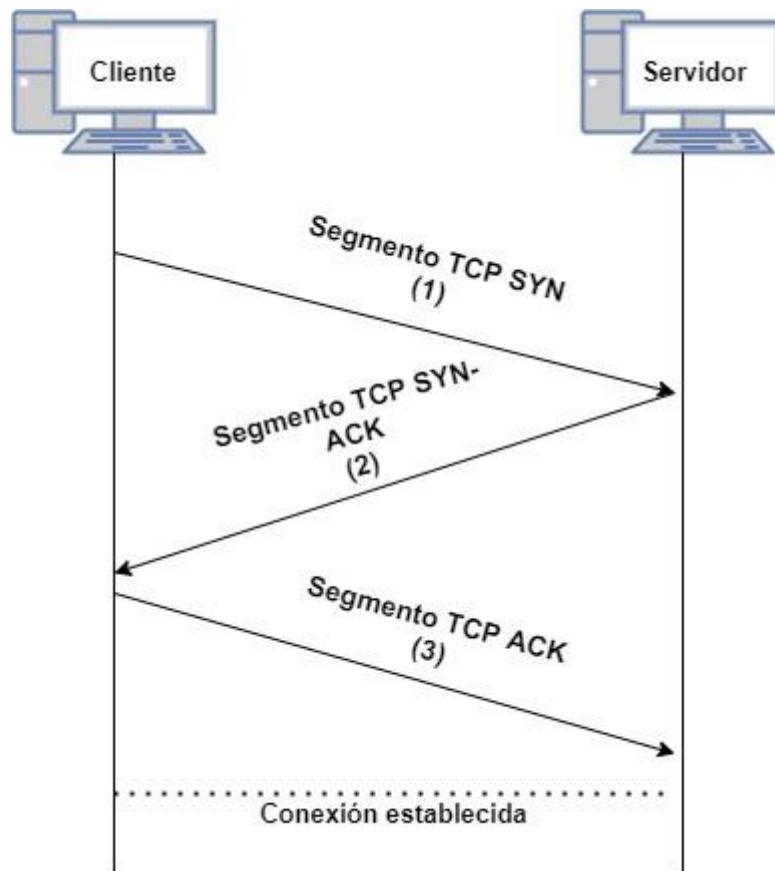


Figura H.0.1 - Establecimiento conexión TCP

En la Figura H.0.1 se pueden ver las tres fases de la que consta el establecimiento de conexión TCP.

- 1) El cliente envía al servidor un segmento TCP con el bit del flag SYN a 1, con este bit a 1 se le indica al servidor que se desea establecer una sesión con él.
- 2) El servidor responde al cliente con un segmento TCP con los bits de los flags SYN y ACK a 1, el flag SYN para indicarle al cliente que queremos establecer una sesión con él, mientras que el bit ACK se usa para asentir el mensaje que ha recibido del cliente en la fase 1.
- 3) El cliente responde al segmento TCP SYN-ACK que recibe del servidor con un segmento TCP ACK para dar el asentimiento de este último.
- 4) El servidor lo recibe y queda establecida así la conexión para poder intercambiar datos.

Cierre de la conexión TCP

La fase de cierre de conexión es parecida a la vista para la apertura, con la salvedad de que cambian algunos flags de los segmentos enviados.

En la Figura H.0.2, se pueden observar los pasos para el cierre de la conexión TCP.

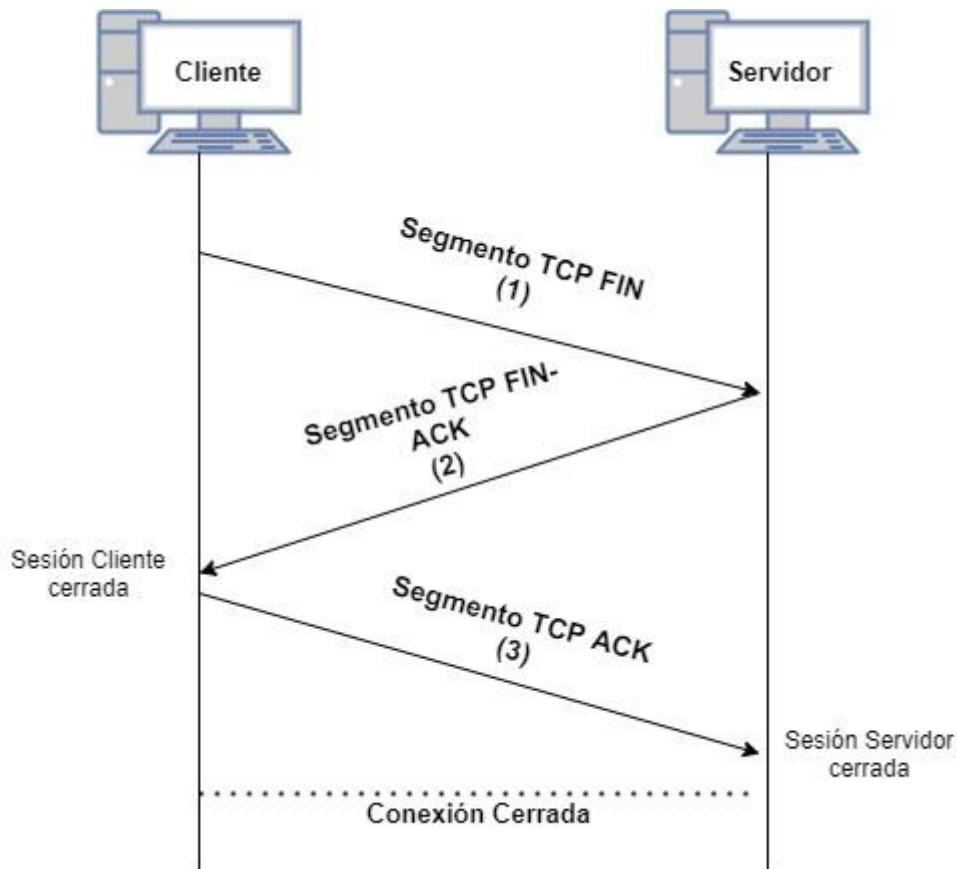


Figura H.0.2 - Cierre conexión TCP

- 1) El cliente envía al servidor un segmento TCP con el bit del flag FIN a 1, con este bit a 1 se le indica al servidor que se desea cerrar la sesión que se tiene establecida con él.
- 2) El servidor responde al cliente con un segmento TCP con los bits de los flags FIN y ACK a 1. El flag FIN indica al cliente que queremos cerrar la sesión que está establecida con él, mientras que el flag ACK se usa para asentir el mensaje que ha recibido del cliente en la fase 1.
- 3) El cliente responde al segmento TCP FIN-ACK que recibe del servidor con un segmento TCP ACK para dar el asentimiento de este último, en este punto el cliente cierra su sesión.
- 4) El servidor lo recibe y cierra su sesión, quedando así cerrada la conexión TCP.

Para más información del protocolo, puede consultarse su referencia (RFC 1323).

Protocolo UDP (User Datagram Protocol)

UDP son las siglas de “Protocolo de Datagramas de Usuario”, el cual se encuentra definido en la RFC 768 y su unidad básica recibe el nombre de datagrama.

Este protocolo opera en el nivel 4 del modelo TCP/IP, basándose para ello en el protocolo IP de capa 3.

UDP a diferencia de TCP, no es un protocolo orientado a conexión, puesto que no necesita realizar una fase de establecimiento de conexión, sino que puede realizar directamente el envío de datos.

Este protocolo tampoco ofrece ni la confirmación de paquetes ni el control de flujo que si proporciona TCP, por lo que los mecanismos de seguridad tienen que ser ofrecidos por protocolos de otras capas.

Este protocolo es usado, sobre todo, en entornos en los que los requisitos de retardo son muy estrictos, como pueden ser flujos de video o audio en streaming, entre otros.

Los protocolos más conocidos de capa de Aplicación que se basan en UDP son: DHCP y DNS (existen más protocolos, pero indicamos los dos más usados).

A continuación, se verá la cabecera usada por UDP.

Cabecera UDP

0		1														2														3																							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31																						
Puerto origen																				Puerto destino																																	
Longitud mensaje																				Checksum																																	

Tabla 35 - Cabecera UDP

En la Tabla 35, se puede observar la composición de una cabecera UDP, a continuación, se dará una breve explicación de cada campo, los campos con un color de fondo distinto representan campos opcionales de UDP.

- **Puerto origen:** Este campo es optativo, e identifica el puerto del proceso emisor (16 bits).
- **Puerto destino:** Identifica el puerto del proceso receptor (16 bits).
- **Longitud del mensaje:** Identifica el tamaño en octetos del mensaje, incluyendo la cabecera UDP y el campo datos (16 bits).
- **Checksum:** Este campo es optativo y se usa para comprobar la integridad del mensaje, tenemos que hacer una apreciación y es que este checksum, usa para calcularse, campos de la cabecera UDP y de la cabecera IP (16 bits).

UDP carece de flags , por lo que no existe distinción entre tipos de mensajes.

Para más información del protocolo puede consultarse su referencia (RFC 768).

Protocolo ICMP (Internet Control Message Protocol)

ICMP son las siglas de “Protocolo de Control de Mensajes de Internet”, el cual se encuentra definido en la RFC 792.

Este protocolo opera en el nivel 3 del modelo TCP/IP, aunque también se apoya en el protocolo Ip que pertenece a su misma capa.

Este protocolo a diferencia de TCP y UDP no busca intercambiar información entre aplicaciones de usuarios, ya que su cometido es el de gestionar los mensajes de error cuando estos ocurren sobre IP.

En este apartado se verá la cabecera usada por ICMP según el tipo de mensaje, así como los mensajes más comunes que podemos encontrar en este protocolo.

Cabecera ICMP

Antes de pasar a ver los tipos de cabecera se realizará una breve introducción al significado de los campos de estas.

- **Tipo ICMP:** Identifica el tipo de mensaje, existen 18 tipos de mensajes, los cuales pueden verse reflejados en la Tabla 36. (8 bits)
- **Código ICMP:** Dentro de un tipo de mensaje ICMP el código sirve para especificar un tipo de mensaje en concreto, los diferentes valores del código pueden comprobarse en la Tabla 36. (8bits)
- **Checksum:** Suma de comprobación para comprobar la integridad del mensaje. (16 bits)
- **Identificador ICMP:** Identifica un flujo de transmisión ICMP, sirve para poder asociar las preguntas con las respuestas (16 bits).
- **Número de Secuencia:** Identifica la posición del mensaje dentro de un flujo con un identificador ICMP concreto (16 bits).

Destination Unreachable

0								1								2								3							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Tipo ICMP								Código ICMP								Checksum															
Identificador ICMP																Número de secuencia															

- **Tipo ICMP:** En este tipo de mensajes, el valor de este campo es siempre 3.
- **Código ICMP:** Puede tomar distintos valores según el tipo de mensaje, estos valores pueden comprobarse en la Tabla 36, yendo desde el valor 0 hasta el 15.

Este tipo de mensajes sirven para avisar de una situación de error, existen situaciones muy diversas en las que estos pueden producirse y cada una de ellas tiene su propio tipo de mensaje, aunque todas tienen en común el

mismo punto y es que no se ha conseguido acceder al destino que se ha solicitado.

Redirect

0							1							2							3										
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Tipo ICMP							Código ICMP							Checksum																	
Sin uso																															

- **Tipo ICMP:** En este tipo de mensajes el valor de este campo es siempre 5.
- **Código ICMP:** Puede tomar distintos valores según el tipo de mensaje, estos valores pueden comprobarse en la Tabla 36 , yendo desde el valor 0 hasta 3.

Este tipo de mensajes es el utilizado por los routers cuando reciben un paquete que deben encaminar y quieren comunicarle al emisor, que existe otra ruta más corta hacia el destino que no pasa por ellos.

Echo-request

0							1							2							3										
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Tipo ICMP							Código ICMP							Checksum																	
Identificador ICMP															Número de secuencia																

- **Tipo ICMP:** En este tipo de mensajes, el valor de este campo es siempre 8.
- **Código ICMP:** En este tipo de mensajes, el valor del campo es siempre 0.

Estos mensajes se usan para solicitar a otro equipo que nos devuelva los mismos datos que nosotros le enviamos previamente.

Echo-reply

0							1							2							3										
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Tipo ICMP							Código ICMP							Checksum																	
Identificador ICMP															Número de secuencia																

- **Tipo ICMP:** En este tipo de mensajes, el valor de este campo es siempre 0.
- **Código ICMP:** En este tipo de mensajes, el valor del campo es siempre 0.

Estos mensajes se usan como respuesta a los mensajes echo-request.

A continuación, se detallará en la Tabla 36 los diferentes valores de tipos y códigos que pueden tener los mensajes ICMP, así como indicar si son mensajes de pregunta o de respuesta.

Tipo	Código	Descripción del Código	Mensaje Pregunta	Mensaje Respuesta
0 - Echo Reply	0	Echo Reply		X
3 - Destination Unreachable	0	Destination network unreachable		X
	1	Destination host unreachable		X
	2	Destination protocol unreachable		X
	3	Destination port unreachable		X
	4	Fragmentation required		X
	5	Source route failed		X
	6	Destination network unknown		X
	7	Destination host unknown		X
	8	Source host isolated		X
	9	Network administratively prohibited		X
	10	Host administratively prohibited		X
	11	Network unreacheable for ToS		X
	12	Host unreacheable for ToS		X
	13	Communication administratively prohibited		X
4 - Source Quench	0	Host precedence Violation		X
	14	Precedence cutoff in effect		X
	15	Source quench		X
5 - Redirect Message	0	Redirect Datagram for the Network		X
	1	Redirect Datagram for the Host		X

	2	Redirect Datagram for the ToS and Network		X
	3	Redirect Datagram for the ToS and Host		X
8 - Echo Request	0	Echo request	X	
11 - Time Exceeded	0	TTL expired in transit		X
	1	Fragment reassembly time exceeded		X
12 - Parameter Problem	0	Pointer indicates the error		X
	1	Missing required option		X
	2	Bad length		X
13 - Timestamp Request	0	Timestamp Request (obsoleto)	X	
14 -Timestamp Reply	0	Timesamp Reply (obsoleto)		X
15 - Information Request	0	Information Request (obsoleto)	X	
16 - Information Reply	0	Information Reply (obsoleto)		X
17- Address mask request	0	Address mask request	X	
18 - Address mask reply	0	Address mask reply		X

Tabla 36 - Tipo mensajes ICMP

Para más información del protocolo, puede consultarse su referencia (RFC 792)

ANEXO I - CONCEPTO CONEXIÓN

En este anexo se estudiará el concepto de conexión, el cuál es necesario entender para poder abordar el capítulo 6, Seguimiento de Conexiones(Conntrack).

Se planteará el concepto de conexión desde el punto de vista de un socket, de un protocolo y de conntrack, explicando que entiende cada uno de ellos por conexión, para finalmente aclarar como debe ser entendida en este punto.

Comenzaremos explicando que es un protocolo orientado a conexión.

Protocolo orientado a conexión.

Existen dos tipos de protocolos, aquellos que están orientados a conexión y los que no lo están.

Un protocolo orientado a conexión es aquel que identifica el flujo de tráfico con un identificador de conexión, en lugar de hacerlo por las direcciones del emisor y el receptor.

Para lograr que se pueda identificar el flujo de conexión, el protocolo establece que antes del envío de datos tendrá lugar una fase de negociación de la comunicación, en la cual, se verificarán datos como la alcanzabilidad, la disponibilidad etc... y una vez se ha dado esta fase inicial y acordados los términos para la transmisión de datos, comenzará la comunicación, identificando todos los paquetes de esta dentro del mismo flujo.

Este flujo de comunicación tiene lugar entre el emisor y el receptor y sirve para identificar a todos los paquetes que se intercambian. (Protocolo TCP (Transmission Control Protocol))

Aquellos protocolos que no realizan esta fase inicial de negociación se conocen como protocolos no orientados a conexión. (Protocolo UDP (User Datagram Protocol))

Socket de conexión.

Un socket es una representación abstracta de un punto de comunicación, que permite establecer un canal de comunicación entre dos rutinas o programas.

Existen dos tipos de sockets, aquellos que son de conexión(sockets de flujo) y los que no son de conexión(socket de datagramas).

Un socket que no está orientado a conexión o socket de datagrama solo guarda su dirección Ip y el puerto en el cual este se encuentra abierto.

Mientras que un socket orientado a conexión o socket de flujo guarda la pareja de datos de Ip/puerto origen/destino.

En este caso los sockets de conexión tienen un concepto parecido al visto en el protocolo orientado a conexión, ya que este busca identificar un flujo de información entre un origen y un destino de los cuales guarda sus datos.

Podemos tener sockets de conexión para TCP y también para UDP.

Debemos entender que al hablar de socket de conexión, lo estamos haciendo a nivel del S.O, es decir, que UDP a pesar de ser un protocolo no orientado a conexión y no establecer una fase preliminar de negociación, si puede a nivel de kernel, emplear un socket de conexión, el cual guarda información del origen y del destino.

Conexión en conntrack

Una vez visto el concepto de conexión de los sockets y de los protocolos, pasaremos a ver que entiende conntrack por conexión.

La definición de conntrack, es módulo de seguimiento de conexiones, esta idea puede llevar a pensar que solamente realiza el seguimiento de protocolos orientados a conexión (TCP) y no es así.

La definición de flujo de conexión para conntrack, no es más que un flujo de datos, es decir, este módulo realiza el seguimiento de un flujo de datos que va desde un origen hasta un destino.

Es por esto, que conntrack muestra entradas de conexión de protocolos como UDP, TCP o ICMP, sin importar si son o no son protocolos orientados a conexión.

Este concepto de conexión es parecido al usado por los sockets, ya que como se ha visto, existen sockets de conexión UDP, que simplemente guardan información de la comunicación entre el origen y el destino.

Pues conntrack se basa en la misma idea.

Cuando llega un paquete, conntrack mira en su cabecera su dirección Ip y puerto, tanto origen como destino y con esos datos es capaz de guardar entradas que realicen el seguimiento de paquetes que pertenezcan a ese flujo de datos.

Un [1]paquete pertenece a un flujo de datos siempre y cuando sus direcciones Ip y puertos origen/destino, coincidan con los que almacene conntrack en sus entradas.

Existe una excepción y es el protocolo ICMP, ya que este carece de puertos, por lo que para realizar el seguimiento de un flujo de datos ICMP, se usa la dirección Ip origen y destino, así como el tipo y el código de ICMP y su Id.

Cada uno de estos protocolos y los datos que conntrack toma de ellos, se verán en profundidad en los puntos dedicados a estos, dentro del capítulo 6.

Conntrack usa otro concepto, el de conexiones relacionadas (RELATED).

Una conexión RELATED, no es más que un flujo de datos que está relacionado con otro flujo que ya se encuentra registrado en el sistema.

Para entender este concepto se puede poner como ejemplo una conexión FTP pasiva.

El flujo de datos de control que se intercambia al principio de la comunicación establece una entrada en conntrack, que tiene IP/puerto origen/destino, pero en el modo pasivo, para el envío de datos se usan puertos distintos, a pesar de que las direcciones Ip son las mismas.

En ese caso y mediante la ayuda de unos módulos llamados helpers (un concepto que se verá en el punto 6.7 Helpers) es capaz de identificar ese nuevo flujo de datos con distinto puerto, como un flujo asociado al ya existente en la máquina y que fue abierto por el intercambio de datos de control.

Una vez explicado, que entiende cada uno de estos puntos por conexión, se puede abordar el capítulo de Seguimiento de Conexiones(Conntrack).

Como resumen final diremos que cuando conntrack hable de seguimiento de conexiones o de flujo de conexión, se estará refiriendo a un flujo de datos o flujo de comunicación.

REFERENCIAS

- [1] **Nftables Web**, <http://nftables.org/> [Online]
- [2] **Iptables project**, <http://www.netfilter.org/projects/iptables/index.html> [Online]
- [3] **Linux Forum**, <https://www.linux.org/> [Online]
- [4] **Ebttables**, <https://ebtables.netfilter.org/> [Online]
- [5] **Arptables man page**, <https://linux.die.net/man/8/arptables> [Online]
- [6] **Wiki Nftables**, https://wiki.nftables.org/wiki-nftables/index.php/Main_Page [Online]
- [7] RedHat packages, <https://access.redhat.com/errata/RHEA-2016:2558> [Online]
- [8] **Shorewall no migration**, <https://sourceforge.net/p/shorewall/mailman/message/35492188/> [Online]
- [9] **Suricata user guide**, <https://fossies.org/linux/suricata/doc/userguide/userguide.pdf>
- [10] **Firewalld with nftables**, <https://developers.redhat.com/blog/2018/08/10/firewalld-the-future-is-nftables/> [Online]
- [11] **Suricata no migration**, <https://sourceforge.net/p/shorewall/mailman/message/35492188/> [Online]
- [12] **Fortinet**, <https://www.fortinet.com/lat> [Online]
- [13] **Cisco Firewall**, https://www.cisco.com/c/es_es/products/security/firewalls/index.html [Online]
- [14] **Zyxel Firewall**, https://www.zyxel.com/es/es/products_services/smb-security_firewalls.shtml?t=c [Online]
- [15] **Palo Alto**, <https://www.paloaltonetworks.es/> [Online]
- [16] **S.O Debian**, <https://www.debian.org/index.es.html> [Online]
- [17] **Paquetes del kernel Ubuntu**, <https://packages.ubuntu.com/bionic/linux-image-4.15.0-129-generic> [Online]
- [18] **Libmnl**, <https://www.netfilter.org/projects/libmnl/doxygen/html/> [Online]

- [19] **Libnftnl**, <https://git.netfilter.org/libnftnl/> [Online]
- [20] **Libgmp**, https://archlinux.org/packages/core/x86_64/gmp/ [Online]
- [21] **Libreadline**, <https://tiswww.case.edu/php/chet/readline/rltop.html> [Online]
- [22] **Familias nftables**, https://wiki.nftables.org/wiki-nftables/index.php/Nftables_families [Online]
- [23] **Netfilter Hooks**, <https://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO-3.html> [Online]
- [24] **Handler nftables**, https://wiki.nftables.org/wiki-nftables/index.php/Simple_rule_management [Online]
- [25] **Pipeline Netfilter**, <http://ftp.gnu.org/pub/doc/packet-journey-2.4.html> [Online]
- [26] **Netfilter hooks**, https://wiki.nftables.org/wiki-nftables/index.php/Netfilter_hooks [Online]
- [27] **Packets in kernel**, <https://blog.packagecloud.io/eng/2017/02/06/monitoring-tuning-linux-networking-stack-sending-data/> [Online]
- [28] **Ipfire nftables**, <https://community.ipfire.org/t/new-features-ipfire-roadmap/4150/7> [Online]
- [29] **Ferm nftables**, <http://ferm.foo-projects.org/download/2.6/NEWS> [Online]
- [30] **Zonas Firewalld**, <https://docs.bluehosting.cl/tutoriales/servidores/introduccion-a-firewalld-en-centos.html> [Online]
- [31] **DNS reference**, <https://tools.ietf.org/html/rfc881> [Online]
- [32] **Principios HTTP**, <http://bibing.us.es/proyectos/abreproy/11214/fichero/TOMO+I%252F05+Capitulo+5+Protocolo+HTTP.pdf> [Online]
- [33] **Project Apache**, <https://httpd.apache.org/> [Online]
- [34] **SSH reference**, <https://tools.ietf.org/html/rfc4250> [Online]
- [35] **Man conntrack**, <https://www.systutorials.com/docs/linux/man/8-conntrack/> [Online]
- [36] **Stateful Firewall**, [https://wiki.archlinux.org/index.php/Simple_stateful_firewall_\(Espa%C3%B1ol\)](https://wiki.archlinux.org/index.php/Simple_stateful_firewall_(Espa%C3%B1ol)) [Online]

- [37] **Máquina de estados**, <https://www.frozenthux.net/iptables-tutorial/spanish/chunkyhtml/c694.html> [Online]
- [38] **FTP reference**, <https://tools.ietf.org/html/rfc959> [Online]
- [39] **Notrack nftables**, https://wiki.nftables.org/wiki-nftables/index.php/Setting_packet_connection_tracking_maintainformation [Online]
- [40] **Entrada conntrack**, <https://www.frozenthux.net/iptables-tutorial/spanish/chunkyhtml/x724.html> [Online]
- [41] **Conntrack helpers**, <https://github.com/regit/secure-conntrack-helpers/blob/master/secure-conntrack-helpers.rst> [Online]
- [42] **Explicación del conntrack**, <http://people.netfilter.org/pablo/docs/login.pdf> [Online]
- [43] **DCCP reference**, <https://tools.ietf.org/html/rfc4340> [Online]
- [44] **SCTP reference**, <https://tools.ietf.org/html/rfc4960> [Online]
- [45] **Nf_conn structure**, https://docs.huihoo.com/doxygen/linux/kernel/3.7/structnf__conn.html [Online]
- [46] **Proc/net/conntrack deprecated**, <https://askubuntu.com/questions/266991/in-ubuntu-12-10-how-to-enable-proc-net-ip-conntrack> [Online]
- [47] **Conntrack-Tools Web**, <http://conntrack-tools.netfilter.org/downloads.html> [Online]
- [48] **Manual Conntrack-tools**, <https://conntrack-tools.netfilter.org/manual.html> [PDF - Online]
- [49] **TCP reference**, <https://tools.ietf.org/html/rfc793> [Online]
- [50] **Estados TCP**, https://code.woboq.org/linux/linux/include/uapi/linux/netfilter/nf_conntrack_tcp.h.html [Online]
- [51] **Herramienta Hping3**, <https://www.blackploit.com/2009/11/manual-hping.html> [Online]
- [52] **Man curl**, <https://curl.se/docs/manpage.html> [Online]
- [53] **UDP reference**, <https://tools.ietf.org/html/rfc768> [Online]
- [54] **ICMP reference**, <https://tools.ietf.org/html/rfc792> [Online]

- [55] **Conntrack code**, https://stuff.mit.edu/afs/sipb/project/merakidev/include/linux/netfilter_ipv4/ip_conntrack.h [Online]
- [56] **Howto NAT**, <https://www.netfilter.org/documentation/HOWTO/NAT-HOWTO.html> [Online]
- [57] **Howto Masquerade**, https://tldp.org/HOWTO/html_single/Masquerading-Simple-HOWTO/ [Online]
- [58] **Stateless NAT**, [https://wiki.nftables.org/wiki-nftables/index.php/Performing_Network_Address_Translation_\(NAT\)](https://wiki.nftables.org/wiki-nftables/index.php/Performing_Network_Address_Translation_(NAT)) [Online]
- [59] **Stateful NAT**, <https://wpollock.com/AUnixSec/IptablesOverview.htm> [Online]
- [60] **Sets nftables**, [http://wiki.nftables.org/wiki-nftables/index.php/Sets](https://wiki.nftables.org/wiki-nftables/index.php/Sets) [Online]
- [61] **Stateful Objects**, <https://marc.info/?l=netfilter-devel&m=148029128323837&w=2> [Online]
- [62] **Port-knocking**, https://es.wikipedia.org/wiki/Golpeo_de_puertos [Online]
- [63] **Mapas nftables**, <https://wiki.nftables.org/wiki-nftables/index.php/Maps> [Online]