Ataques ICMP contra TCP

Fernando Gont

UTN/FRH, Argentina

Jornada de Seguridad en Internet

15 de agosto de 2007, Montevideo, Uruguay

Aislamiento de fallas y recobro de fallas en la Arquitectura de Internet

Dos funciones básicas en una red de computadoras son el **aislamiento de fallas** y el **recobro de fallas**.

- El aislamiento de fallas consiste en detectar condiciones de error en la red (sistemas inalcanzables, etc.)
- El recobro de fallas consiste en intentar sobrevivir esas condiciones de error.
- La Arquitectura de Internet delega la función de aislamiento de fallas al protocolo ICMP.

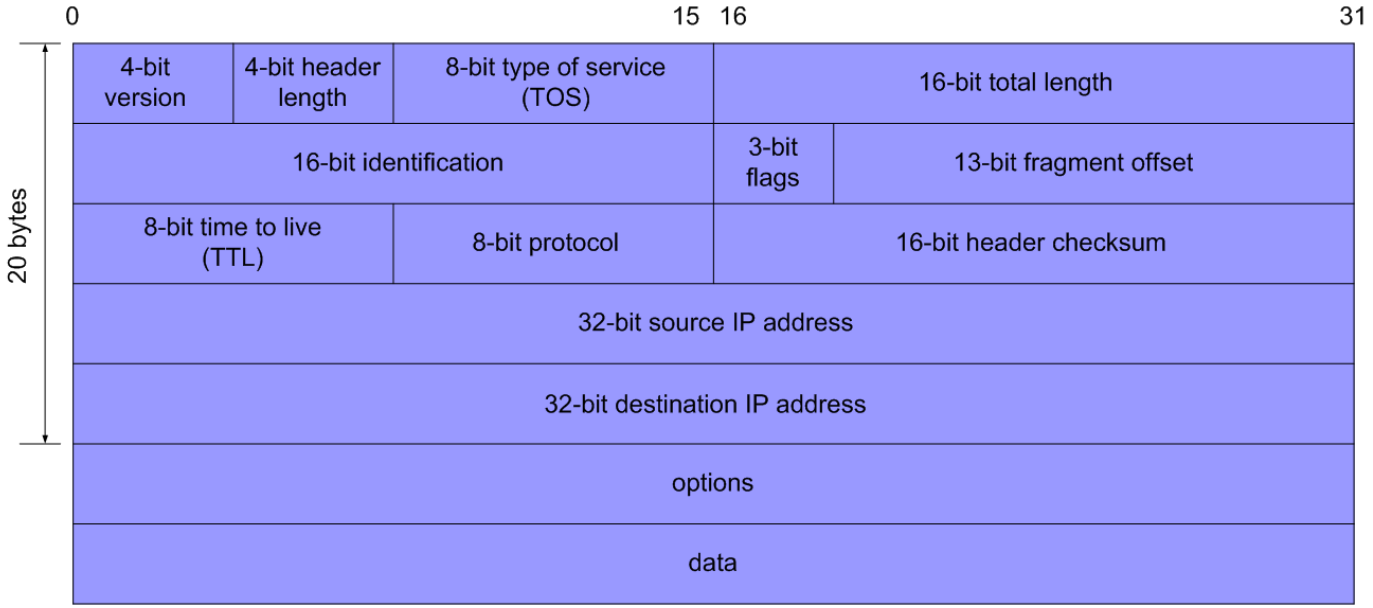
Internet Control Message Protocol

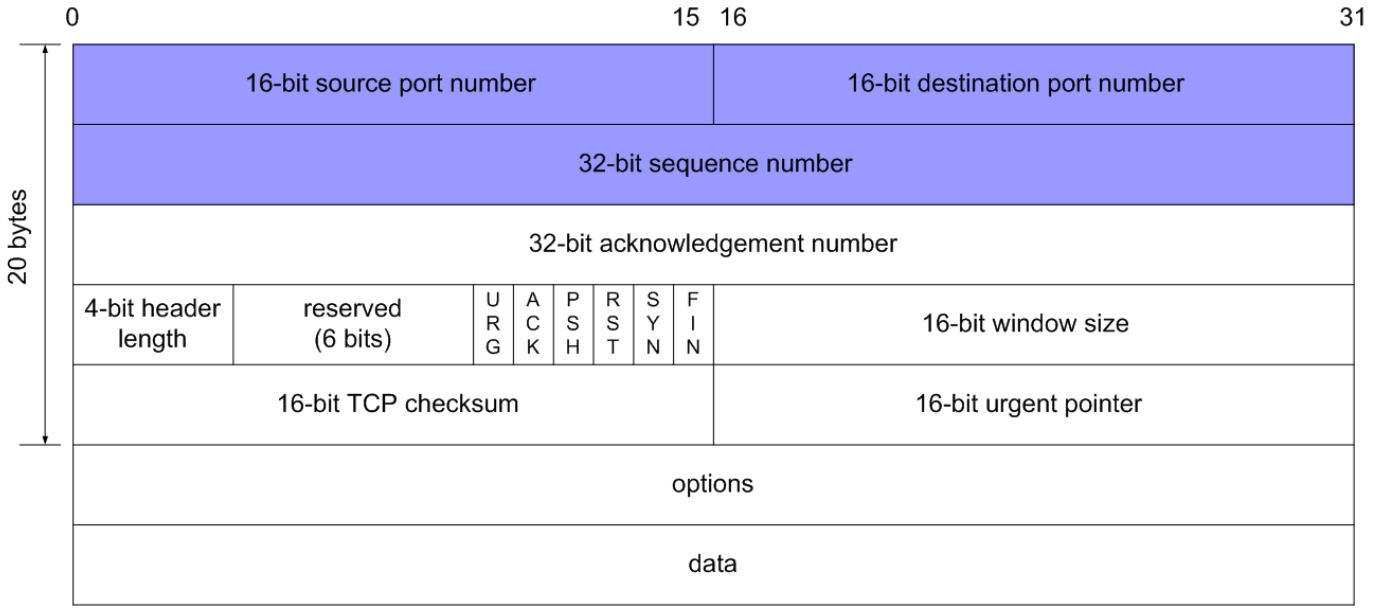
- Conceptualmente, el protocolo ICMP forma parte de la capa de red, trabajando conjuntamente con el protocolo IP.
- Se utiliza principalmente para la señalización de errores en la capa de red (IP)
- Asimismo, también se utiliza para realizar algunas funciones de depuración de errores de red. Herramientas como ping y traceroute basan su funcionamiento en el protocolo ICMP.
- Originalmente, también se lo utilizaba para algunas funciones de configuración de sistemas (obtención de máscara de red, obtención de dirección de router, etc.)

Demultiplexación de mensajes ICMP

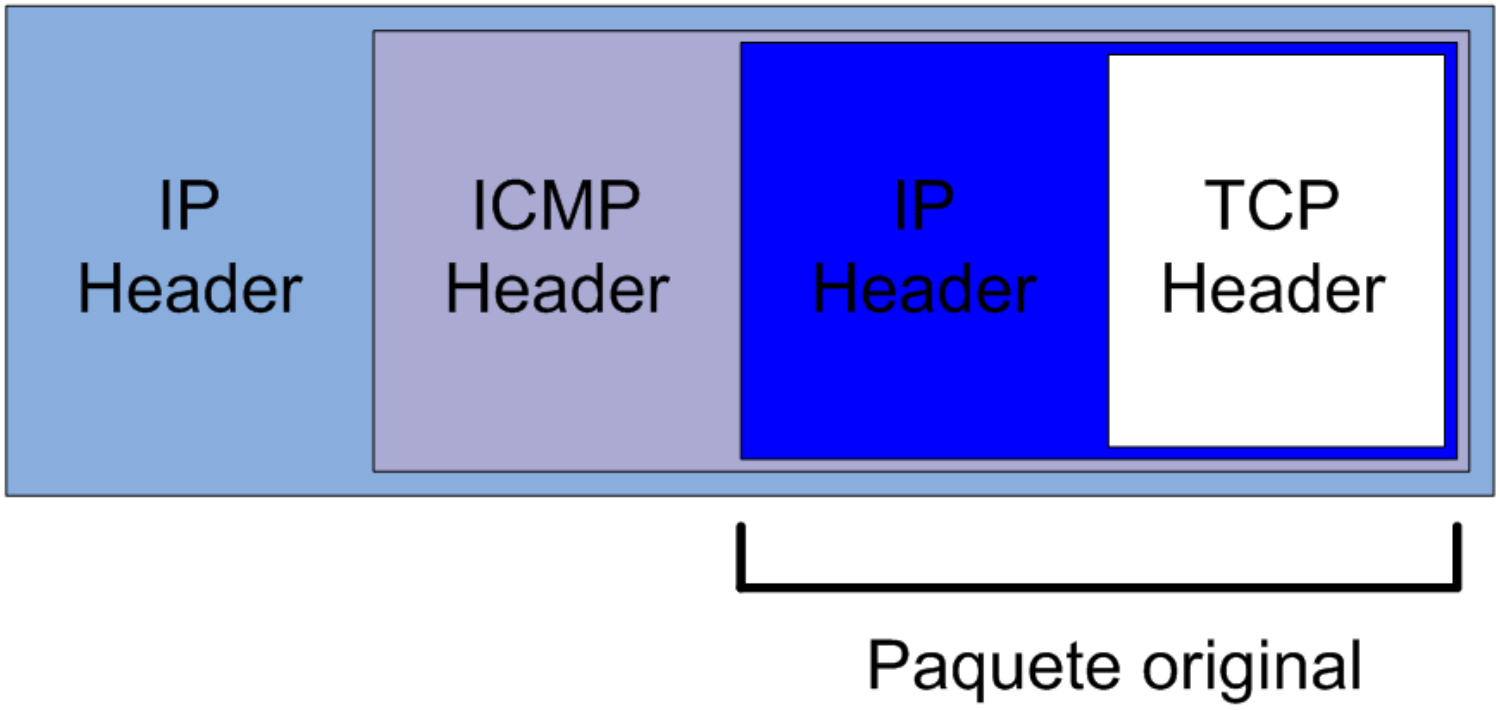
- Cuando un sistema recibe un mensaje ICMP, se desea notificar dicho evento a la instancia de comunicación que lo produjo, con el fin de que el correspondiente protocolo de transporte realice su operación de “recobro de fallos”.
- Con el fin de posibilitar esta operación, los mensajes de error ICMP incluirán una porción del paquete que causó el error en cuestión, bajo la suposición que dicha porción del paquete original proveerá toda la información necesaria para demultiplexar el mensaje de error.
- Las especificaciones de la IETF requieren que se incluya en el mensaje de error ICMP al menos el encabezado IP completo y los primeros 8 bytes del IP payload del paquete que generó el mensaje de error.

Información incluída en los mensajes ICMP





Estructura del paquete resultante



El paquete ICMP contiene en su “payload” parte del paquete original que causó el error. Dicho paquete ICMP se encapsula en un paquete IP, para ser enviado hacia el sistema que debe recibir el mensaje de error.

Validación de mensajes ICMP

Las especifiaciones de la IETF no recomiendan ningún tipo de chequeo en los mensajes de error ICMP recibidos.

Esto significa que siempre que el mensaje de error ICMP contenga los valores {source IP, source TCP port, destination IP, destination TCP port} correctos, será pasado a la correspondiente instancia de TCP, y será procesado, causando las acciones recomendadas por las especificaciones

Información necesaria para atacar

Para lograr que un mensaje de error ICMP sea entregadoo a una instancia de TCP dada, el atacante tendría que "adivinar":

- Dirección IP origen (Source IP)
- Dirección IP destino (Destination IP)
- Puerto TCP origen (Source TCP port)
- Puerto TCP destino (Destination TCP port)

En principio, esto haría suponer que la información necesaria para realizar ataques mediante la falsificación de mensajes ICMP es demasiada como para que los ataques sean realizables en la práctica.

Sin embargo, no hay tanto que adivinar....

- La dirección IP del servidor usualmente será conocida
- La dirección IP del cliente podría ser conocida
- El puerto TCP del servidor usualmente será conocido
- El puerto TCP del cliente usualmente no será conocido, pero podrá ser adivinado

Asumiendo que el atacante conoce las partes involucradas, y el servicio utilizado por las mismas,

Cuanto mucho, el atacante deberá enviar 65536 paquetes para realizar cualquier tipo de ataque ICMP contra TCP

Rango de puertos efímeros

La mayoría de los sistemas eligen sus “puertos efímeros” de un subespacio de todo el espacio de puertos disponible

Sistema operativo	Puertos efímeros
Microsoft Windows	1024 - 4999
Linux kernel 2.6	1024 - 4999
Solaris	32768 - 65535
AIX	32768 - 65535
FreeBSD	1024 - 5000
NetBSD	49152 - 65535
OpenBSD	1024 - 65535

Número práctico de paquetes requeridos para realizar un ataque ICMP contra TCP

En la mayoría de los casos, el atacante precisará enviar **a lo sumo** 4K paquetes para realizar cualquier ataque ICMP contra TCP

Con un enlace de 128 kbps, esto llevaría nada más que **unos pocos segundos**!

El eslabón mas débil de la cadena

- Ninguna de las contra-medidas existentes para proteger a TCP de otros ataques ayudan a proteger TCP de los ataques ICMP
- Los mensajes ICMP pueden provenir desde cualquier sistema de Internet. Es imposible predecir qué router encontrará una condición de error, por lo cual es imposible predecir qué sistema enviará un mensaje de error ICMP. Ásímismo, un sistema podría tener más de una dirección IP, y enviar el mensaje de error ICMP utilizando cualquiera de ellas.
- Esto implica que no hace falta falsificar la dirección de origen de los mensajes ICMP, siquiera.

Todas estas consideraciones hacen que los ataques ICMP sean los mas simples de realizar contra TCP y otros protocolos de transporte similares

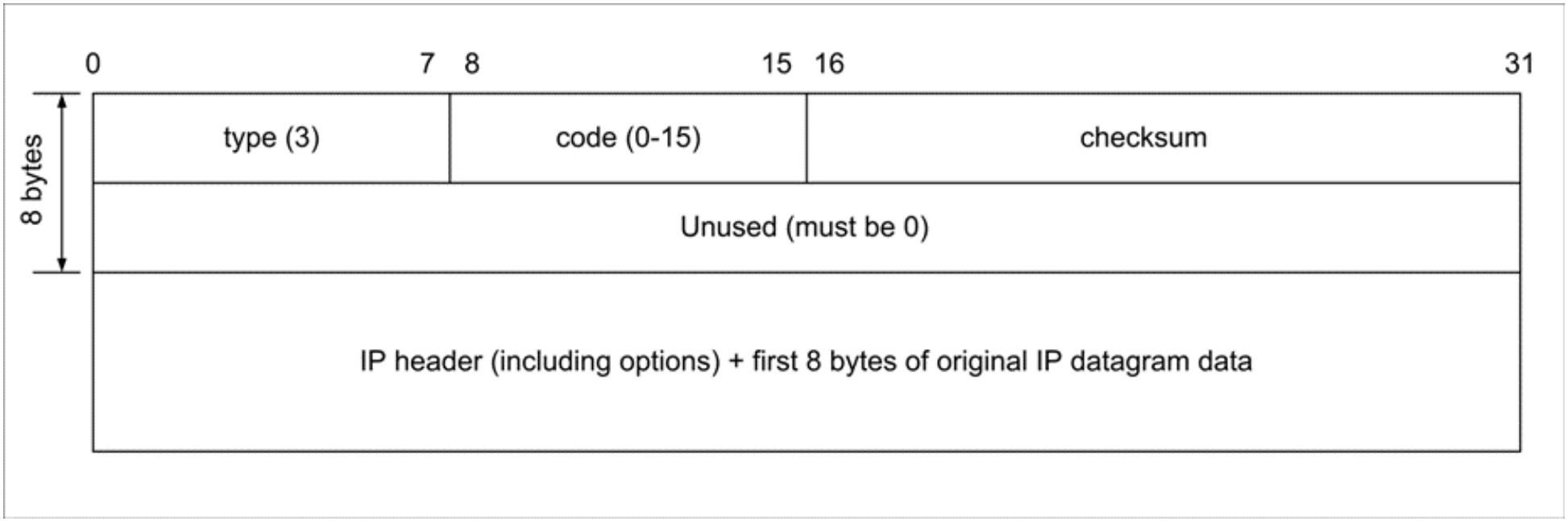
Blind connection-reset attack

- Reseteando ciegamente una conexión TCP arbitraria

Generación de mensajes ICMP

- Cuando un sistema detecta una condición de error en la red, usualmente emitirá un mensaje de error ICMP, para notificar sobre la condición de error al sistema remitente del paquete que la generó.
- ICMP define una serie de mensajes de error, para notificar diversas condiciones de error en la red.
- Algunos de estos mensajes pueden ser generados por sistemas intermediarios (routers), mientras que otros pueden ser generados por sistemas finales (hosts).

Formato de los mensajes de error ICMP



El formato de los mensajes ICMP es básicamente el mismo para todos los valores de “type” y “code”

Mensajes de error ICMP

Type	Code	Descripción
3	0	Net unreachable
3	1	Host unreachable
3	2	Protocol Unreachable
3	3	Port unreachable
3	4	Frag. needed and DF set
3	5	Source route failed

Clasificación de mensajes de error ICMP

Las especificaciones de la IETF hacen una clasificación grosera de los mensajes de error ICMP en aquellos que se suponen indicar “**errores leves**” (soft errors), y aquellos que suponen indicar “**errores groseros**” (hard errors)

- Los errores leves son aquellos que se supone que se solucionarán en un corto plazo.
- Los errores groseros son aquellos que se supone que no desaparecerán en el corto plazo.

Clasificación de mensajes de error ICMP

Code	Descripción	Clasificación
0	Net unreachable	leve
1	Host unreachable	leve
2	Protocol Unreachable	grave
3	Port unreachable	grave
4	Frag. needed and DF set	grave
5	Source route failed	Leve

Reacción de TCP a los mensajes ICMP

Cuando TCP es notificado de una condición de error, el mismo aplicará su política de recobro de fallos:

- Si el error notificado es un “error leve” (soft error), TCP recordará el error, y continuará retransmitiendo la información hasta recibir un acuse de recibo, o hasta que decida abortar la conexión (por haber reintentado muchas veces).
- Si el error notificado es un “error grave” (hard error), TCP abortará la correspondiente conexión inmediatamente.

Blind connection-reset attack

Un atacante podría falsificar un mensaje de error ICMP que indique un “error grosero” (hard error).

Como resultado, tal como indica el RFC 1122,

“TCP SHOULD abort the connection”

(*“TCP DEBERÍA abortar la conexión”*)

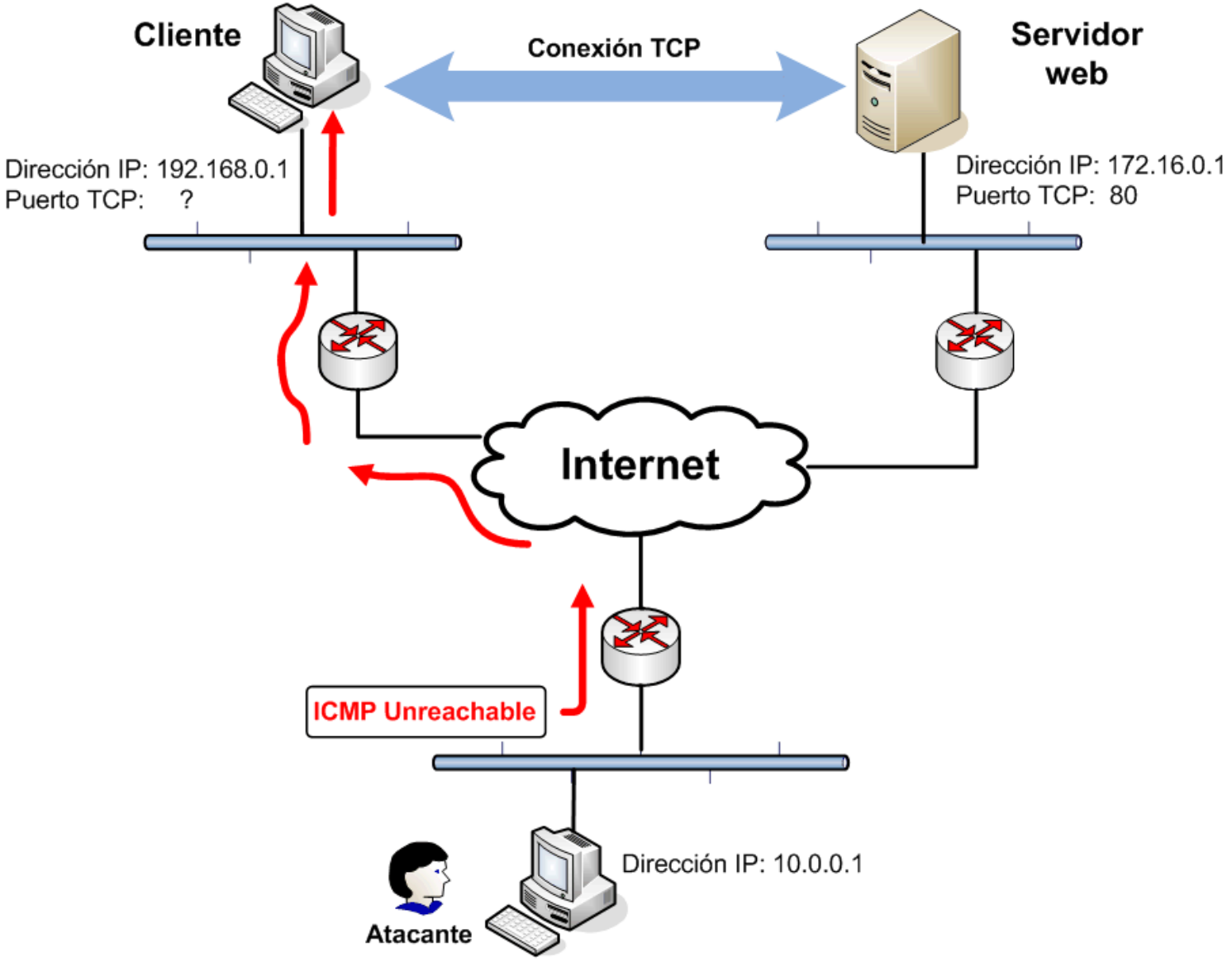
Aplicaciones afectadas

Las aplicaciones mas afectadas serán aquellas que requieren conexiones de larga vida para un correcto funcionamiento. Entre ellas podríamos encontrar:

- BGP
- Sistemas de backup vía-red
- VoIP (si se utiliza TCP para la señalización)

Si el objetivo del ataque es BGP, este ataque podría provocar un estado de negación de servicio a redes enteras

Posible escenario de un ataque



Tcpdump output

22:20:56 172.16.0.1.80 > 192.168.0.1.3270: . 58849:60269(1420) ack 203 win 17040 (DF) (ttl 63, id 36261)
22:20:57 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 51749 win 9940 (DF) (ttl 118, id 23142)
22:20:57 172.16.0.1.80 > 192.168.0.1.3270: . 60269:61689(1420) ack 203 win 17040 (DF) (ttl 63, id 63275)
22:20:57 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 53169 win 9940 (DF) (ttl 118, id 23143)
22:20:57 172.16.0.1.80 > 192.168.0.1.3270: . 61689:63109(1420) ack 203 win 17040 (DF) (ttl 63, id 36878)
22:20:58 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 54589 win 9940 (DF) (ttl 118, id 23144)
22:20:58 172.16.0.1.80 > 192.168.0.1.3270: . 63109:64529(1420) ack 203 win 17040 (DF) (ttl 63, id 55051)
22:20:58 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 56009 win 9940 (DF) (ttl 118, id 23146)
22:20:58 172.16.0.1.80 > 192.168.0.1.3270: . 64529:65949(1420) ack 203 win 17040 (DF) (ttl 63, id 51041)
22:20:58 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 57429 win 9940 (DF) (ttl 118, id 23147)
22:20:58 172.16.0.1.80 > 192.168.0.1.3270: . 65949:67369(1420) ack 203 win 17040 (DF) (ttl 63, id 59428)
22:20:58 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 58849 win 9940 (DF) (ttl 118, id 23148)
22:20:58 172.16.0.1.80 > 192.168.0.1.3270: . 67369:68789(1420) ack 203 win 17040 (DF) (ttl 63, id 56440)
22:20:59 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 60269 win 9940 (DF) (ttl 118, id 23152)
22:20:59 172.16.0.1.80 > 192.168.0.1.3270: . 68789:70209(1420) ack 203 win 17040 (DF) (ttl 63, id 53543)
22:20:59 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 61689 win 9940 (DF) (ttl 118, id 23153)
22:20:59 172.16.0.1.80 > 192.168.0.1.3270: . 70209:71629(1420) ack 203 win 17040 (DF) (ttl 63, id 56257)
22:20:59 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 63109 win 9940 (DF) (ttl 118, id 23154)
22:20:59 172.16.0.1.80 > 192.168.0.1.3270: . 71629:73049(1420) ack 203 win 17040 (DF) (ttl 63, id 43027)
22:20:59 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 64529 win 9940 (DF) (ttl 118, id 23156)
22:20:59 172.16.0.1.80 > 192.168.0.1.3270: . 73049:74469(1420) ack 203 win 17040 (DF) (ttl 63, id 34869)
22:21:00 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 65949 win 9940 (DF) (ttl 118, id 23158)
22:21:00 172.16.0.1.80 > 192.168.0.1.3270: . 74469:75889(1420) ack 203 win 17040 (DF) (ttl 63, id 42831)
22:21:00 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 67369 win 9940 (DF) (ttl 118, id 23159)
22:21:00 172.16.0.1.80 > 192.168.0.1.3270: . 75889:77309(1420) ack 203 win 17040 (DF) (ttl 63, id 38361)
22:21:00 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 68789 win 9940 (DF) (ttl 118, id 23162)
22:21:00 172.16.0.1.80 > 192.168.0.1.3270: . 77309:78729(1420) ack 203 win 17040 (DF) (ttl 63, id 47968)

Tcpdump output (continuado)

```
22:21:00 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 70209 win 9940 (DF) (ttl 118, id 23164)
22:21:00 172.16.0.1.80 > 192.168.0.1.3270: . 78729:80149(1420) ack 203 win 17040 (DF) (ttl 63, id 56881)
22:21:01 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 71629 win 9940 (DF) (ttl 118, id 23165)
22:21:01 172.16.0.1.80 > 192.168.0.1.3270: . 80149:81569(1420) ack 203 win 17040 (DF) (ttl 63, id 50563)
22:21:01 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 74469 win 9940 (DF) (ttl 118, id 23167)
22:21:01 172.16.0.1.80 > 192.168.0.1.3270: . 81569:82989(1420) ack 203 win 17040 (DF) (ttl 63, id 39445)
22:21:01 172.16.0.1.80 > 192.168.0.1.3270: . 82989:84409(1420) ack 203 win 17040 (DF) (ttl 63, id 61324)
22:21:01 172.16.0.1 > 192.168.0.1: icmp: 172.16.0.1 protocol 6 unreachable for 192.168.0.1.3270 > 172.16.0.1.80:
        [|tcp] (ttl 158, id 61654) (ttl 214, id 31456)
22:21:02 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 75889 win 9940 (DF) (ttl 118, id 23169)
22:21:02 172.16.0.1.80 > 192.168.0.1.3270: . 84409:85829(1420) ack 203 win 17040 (DF) (ttl 63, id 46016)
22:21:02 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 78729 win 9940 (DF) (ttl 118, id 23171)
22:21:02 172.16.0.1.80 > 192.168.0.1.3270: . 85829:87249(1420) ack 203 win 17040 (DF) (ttl 63, id 64644)
22:21:02 172.16.0.1.80 > 192.168.0.1.3270: . 87249:88669(1420) ack 203 win 17040 (DF) (ttl 63, id 39376)
22:21:02 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 80149 win 9940 (DF) (ttl 118, id 23173)
22:21:02 172.16.0.1.80 > 192.168.0.1.3270: . 88669:90089(1420) ack 203 win 17040 (DF) (ttl 63, id 58117)
22:21:02 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 81569 win 9940 (DF) (ttl 118, id 23175)
22:21:02 172.16.0.1.80 > 192.168.0.1.3270: . 90089:91509(1420) ack 203 win 17040 (DF) (ttl 63, id 62887)
22:21:03 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 82989 win 9940 (DF) (ttl 118, id 23176)
22:21:03 172.16.0.1.80 > 192.168.0.1.3270: . 91509:92929(1420) ack 203 win 17040 (DF) (ttl 63, id 54586)
22:21:03 192.168.0.1.3270 > 172.16.0.1.80: . [tcp sum ok] 203:203(0) ack 84409 win 9940 (DF) (ttl 118, id 23177)
22:21:03 172.16.0.1.80 > 192.168.0.1.3270: . 92929:94349(1420) ack 203 win 17040 (DF) (ttl 63, id 56692)
22:21:03 192.168.0.1.3270 > 172.16.0.1.80: R [tcp sum ok] 4174694923:4174694923(0) win 0 (ttl 118, id 23180)
```

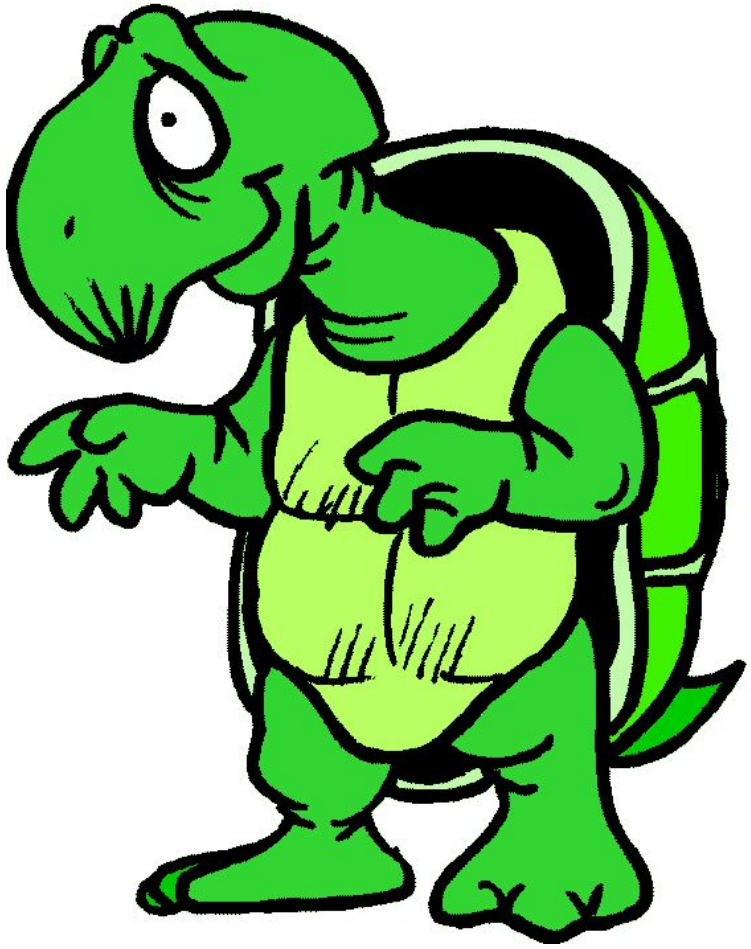
Solucionando la vulnerabilidad

Son los “errores graves” realmente graves?

Si se reporta un supuesto “error grave” a una conexión que **ya ha sido establecida**, entonces el error no puede ser “grave”. Si lo fuera, nunca podría haber establecido la conexión!

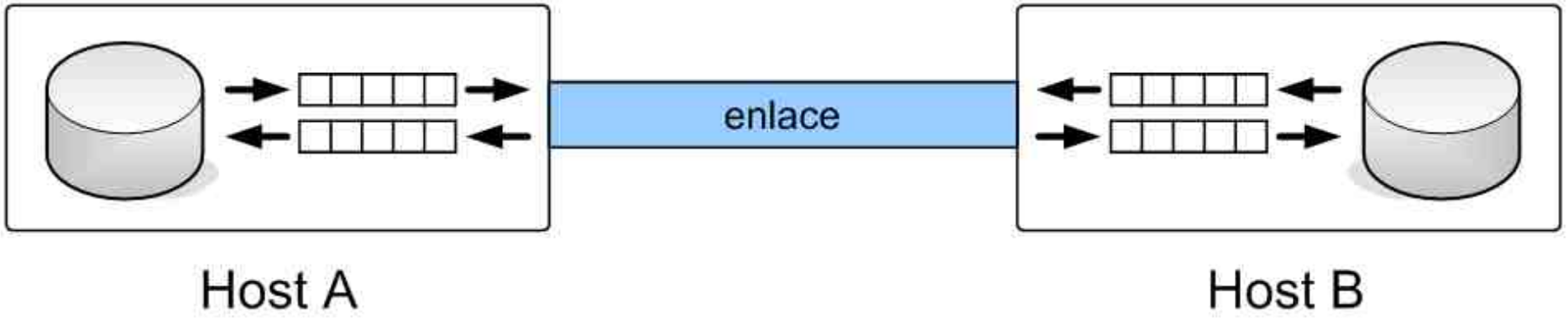
Para conexiones TCP en cualquiera de los estados sincronizados (es decir, desde que se ha establecido la conexión, en adelante), todos los errores ICMP deberían ser considerados “leves”

Blind throughput-reduction attack



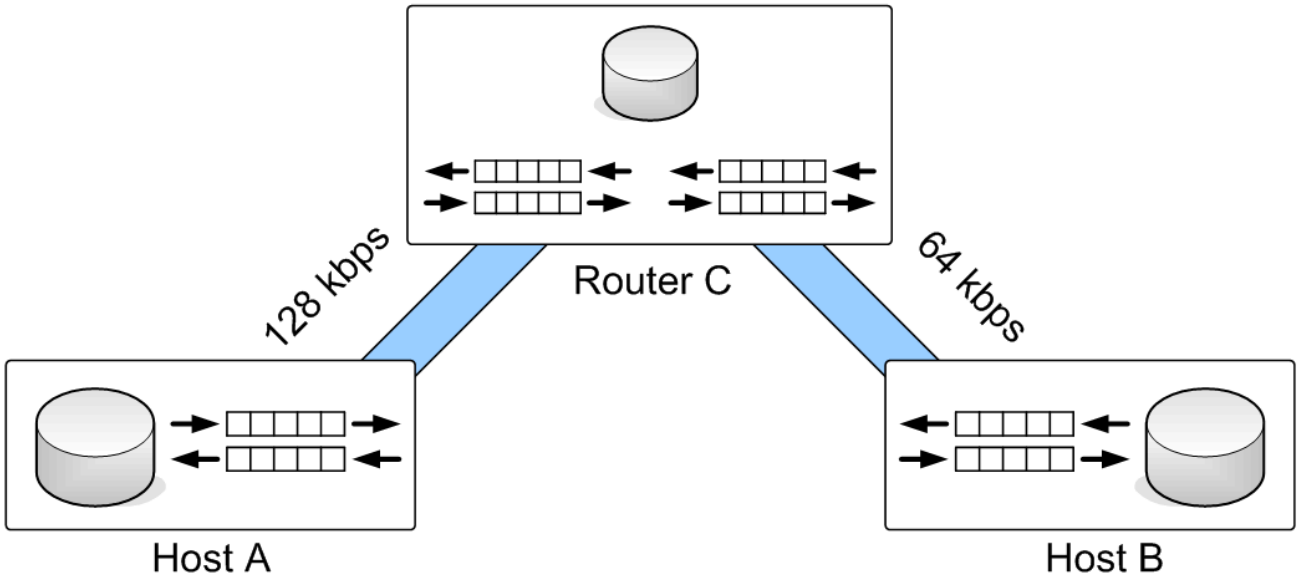
- Reduciendo ciegamente la tasa de transferencia de una conexión TCP arbitraria

Modelo de comunicación entre dos sistemas



- Cada uno de los sistemas contiene una “cola de entrada” y una “cola de salida”.
- El control de flujo apunta a controlar el flujo de información entre dos sistemas, para evitar que un sistema “rápido” sature a otro más “lento”

Modelo de comunicación en una red de conmutación de paquetes

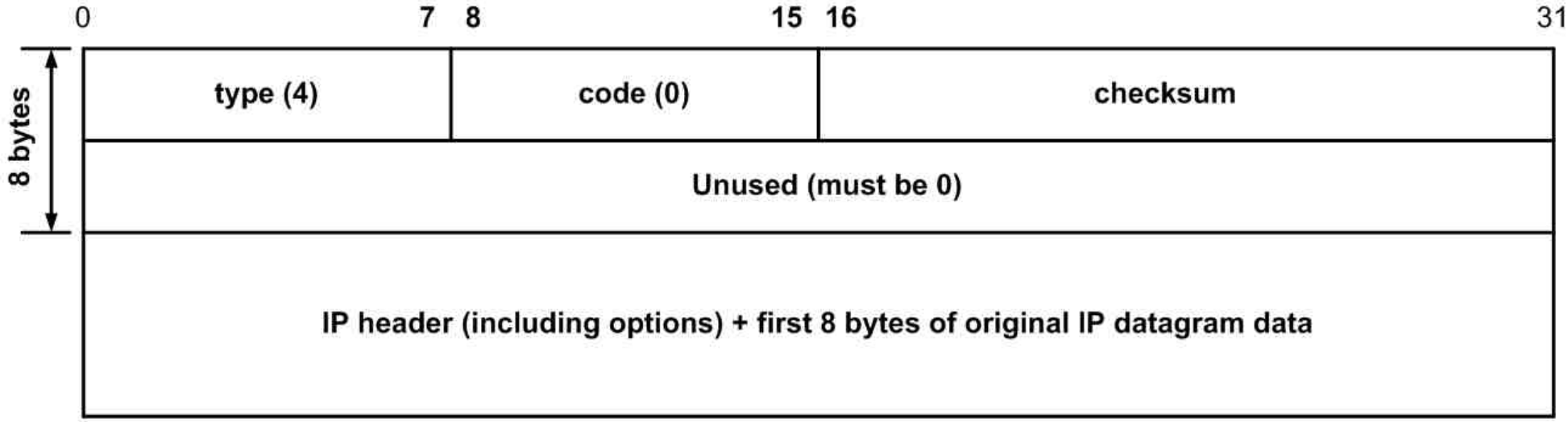


- El control de congestión apunta a controlar el efecto producido por el **conjunto** de flujos de información que toman lugar en una red, para evitar la congestión de la misma.
- El hecho de que un flujo individual esté controlado, no implica que el **conjunto** de los distintos flujos no impacte negativamente el funcionamiento de una red.

Control de congestión en la Arquitectura de Internet

- El control de congestión reside típicamente en la capa de red (network layer).
- Sin embargo, la Arquitectura de Internet provee escaso soporte en la capa de red para el control de congestión.
- El único mecanismo provisto es el mensaje ICMP Source Quench (un paquete de choque).

Formato del mensaje ICMP Source Quench



- El mensaje ICMP Source Quench contendrá el encabezamiento IP completo más los primeros 8 bytes del IP payload, con el fin de que el mismo pueda ser demultiplexado.

Uso del mensaje ICMP Source Quench

Es descripto en el RFC 792 como:

“A gateway may discard internet datagrams if it does not have the buffer space needed to queue the datagrams for output to the next network on the route to the destination Network. If a gateway discards a datagram, it may send a source quench message to the internet source host of the datagram. A destination host may also send a source quench message if datagrams arrive too fast to be processed. The source quench message is a request to the host to cut back the rate at which it is sending traffic to the internet destination.”

Procesamiento de mensajes ICMP Source Quench por parte de TCP

El RFC 792 dice:

"On receipt of a source quench message, the source host should cut back the rate at which it is sending traffic to the specified destination until it no longer receives source quench messages from the gateway."

El RFC 1122 agrega:

"TCP MUST react to a Source Quench by slowing transmission on the connection. The RECOMMENDED procedure is for a Source Quench to trigger a 'slow start,' as if a retransmission timeout had occurred."

Blind throughput-reduction attack

Un atacante podría falsificar mensajes ICMP Source Quench para hacerle creer al sistema que se encuentra enviando información que la red se encuentra congestionada

Impacto del ataque

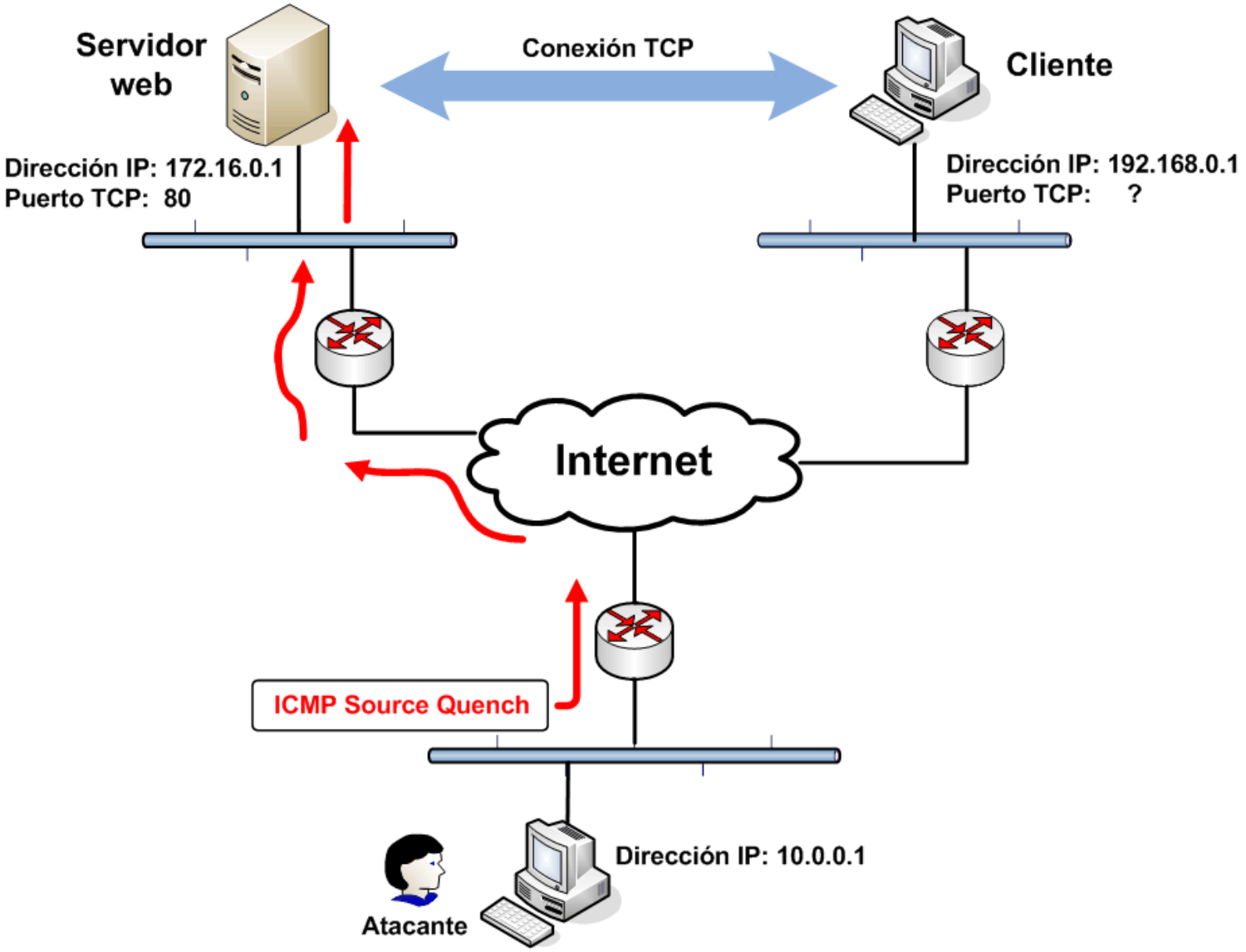
Un atacante podría reducir la tasa de transferencia de una conexión TCP arbitraria, incluso estando fuera del camino seguido por los paquetes pertenecientes a la conexión en cuestión.

El envío contínuo de paquetes ICMP Source Quench podría reducir la tasa de transferencia de una conexión TCP a aproximadamente **1 paquete por RTT** (Round-Trip Time).

Aplicaciones afectadas

Los protocolos de aplicación mas afectados serán aquellos que dependen en conexiones de alta tasa de transferencia para su correcto funcionamiento.

Posible escenario de un ataque



Salida de tcpdump

01:47:56 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 71, id 3721) (ttl 188, id 38453)

01:47:56 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 124, id 15000) (ttl 61, id 34927)

01:47:57 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 229, id 25845) (ttl 250, id 45209)

01:47:57 192.168.0.1.1063 > 172.16.0.1.80: . [tcp sum ok] 232:232(0) ack 291649 win 7312 <nop,nop,timestamp 32232 447226421> (DF) [tos 0xf (EC)] (ttl 116, id 45064)

01:47:57 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 222, id 42066) (ttl 123, id 18038)

01:47:57 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 83, id 45730) (ttl 136, id 41605)

01:47:57 192.168.0.1.1063 > 172.16.0.1.80: . [tcp sum ok] 232:232(0) ack 293097 win 8760 <nop,nop,timestamp 32235 447226421> (DF) [tos 0xf (EC)] (ttl 116, id 45320)

01:47:57 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 104, id 26156) (ttl 85, id 48347)

01:47:57 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 249, id 14568) (ttl 238, id 44119)

01:47:57 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 242, id 49311) (ttl 151, id 10965)

01:47:57 192.168.0.1.1063 > 172.16.0.1.80: . [tcp sum ok] 232:232(0) ack 294545 win 7312 <nop,nop,timestamp 32238 447226423> (DF) [tos 0xf (EC)] (ttl 116, id 45576)

01:47:57 172.16.0.1.80 > 192.168.0.1.1063: . 295993:297441(1448) ack 232 win 17376 <nop,nop,timestamp 447226426 32238> (DF) (ttl 64, id 16156)

01:47:57 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 63, id 14055) (ttl 108, id 3600)

01:47:57 192.168.0.1.1063 > 172.16.0.1.80: . [tcp sum ok] 232:232(0) ack 295993 win 8760 <nop,nop,timestamp 32238 447226423> (DF) [tos 0xf (EC)] (ttl 116, id 45832)

01:47:57 172.16.0.1.80 > 192.168.0.1.1063: . 297441:298889(1448) ack 232 win 17376 <nop,nop,timestamp 447226426 32238> (DF) (ttl 64, id 4839)

Salida de tcpdump *(cont.)*

01:47:57 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 232, id 61992) (ttl 161, id 53393)

01:47:58 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 161, id 60570) (ttl 98, id 2081)

01:47:58 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 118, id 15382) (ttl 171, id 61130)

01:47:58 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 131, id 39528) (ttl 116, id 55998)

01:47:58 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 136, id 57047) (ttl 249, id 50387)

01:47:58 192.168.0.1.1063 > 172.16.0.1.80: . [tcp sum ok] 232:232(0) ack 297441 win 7312 <nop,nop,timestamp 32245 447226426> (DF) [tos 0xf (EC)] (ttl 116, id 47368)

01:47:58 172.16.0.1.80 > 192.168.0.1.1063: . 298889:300337(1448) ack 232 win 17376 <nop,nop,timestamp 447226427 32245> (DF) (ttl 64, id 8494)

01:47:58 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 77, id 32815) (ttl 174, id 50351)

01:47:58 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 66, id 6352) (ttl 187, id 37417)

01:47:58 192.168.0.1.1063 > 172.16.0.1.80: . [tcp sum ok] 232:232(0) ack 298889 win 8760 <nop,nop,timestamp 32247 447226426> (DF) [tos 0xf (EC)] (ttl 116, id 47624)

01:47:58 172.16.0.1.80 > 192.168.0.1.1063: . 300337:301785(1448) ack 232 win 17376 <nop,nop,timestamp 447226427 32247> (DF) (ttl 64, id 28561)

01:47:58 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 195, id 57048) (ttl 144, id 1692)

01:47:58 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 152, id 10915) (ttl 169, id 39043)

01:47:58 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 109, id 62567) (ttl 166, id 57565)

01:47:59 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 122, id 21511) (ttl 199, id 5731)

Salida de tcpdump *(cont.)*

01:47:59 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 139, id 650) (ttl 72, id 37585)

01:47:59 192.168.0.1.1063 > 172.16.0.1.80: . [tcp sum ok] 232:232(0) ack 300337 win 7312 <nop,nop,timestamp 32252 447226427> (DF) [tos 0xf (EC)] (ttl 116, id 48136)

01:47:59 172.16.0.1.80 > 192.168.0.1.1063: . 301785:303233(1448) ack 232 win 17376 <nop,nop,timestamp 447226429 32252> (DF) (ttl 64, id 13370)

01:47:59 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 96, id 54804) (ttl 97, id 19491)

01:47:59 192.168.0.1.1063 > 172.16.0.1.80: . [tcp sum ok] 232:232(0) ack 301785 win 8760 <nop,nop,timestamp 32254 447226427> (DF) [tos 0xf (EC)] (ttl 116, id 48392)

01:47:59 172.16.0.1.80 > 192.168.0.1.1063: . 303233:304681(1448) ack 232 win 17376 <nop,nop,timestamp 447226429 32254> (DF) (ttl 64, id 16919)

01:47:59 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 97, id 18598) (ttl 206, id 23717)

01:47:59 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 194, id 61461) (ttl 111, id 31369)

01:47:59 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 103, id 1646) (ttl 152, id 32425)

01:47:59 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 180, id 58070) (ttl 205, id 57556)

01:47:59 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 229, id 31980) (ttl 206, id 14368)

01:47:59 192.168.0.1.1063 > 172.16.0.1.80: . [tcp sum ok] 232:232(0) ack 303233 win 7312 <nop,nop,timestamp 32259 447226429> (DF) [tos 0xf (EC)] (ttl 116, id 48904)

01:47:59 172.16.0.1.80 > 192.168.0.1.1063: . 304681:306129(1448) ack 232 win 17376 <nop,nop,timestamp 447226430 32259> (DF) (ttl 64, id 1204)

01:48:00 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 254, id 60533) (ttl 211, id 48718)

01:48:00 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 143, id 59926) (ttl 248, id 51853)

Salida de tcpdump *(cont.)*

01:48:00 192.168.0.1.1063 > 172.16.0.1.80: . [tcp sum ok] 232:232(0) ack 304681 win 8760 <nop,nop,timestamp 32261 447226429> (DF) [tos 0xf (EC)] (ttl 116, id 49160)

01:48:00 172.16.0.1.80 > 192.168.0.1.1063: . 306129:307577(1448) ack 232 win 17376 <nop,nop,timestamp 447226430 32261> (DF) (ttl 64, id 15196)

01:48:00 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 132, id 20147) (ttl 89, id 21594)

01:48:00 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 221, id 37084) (ttl 134, id 29832)

01:48:00 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 238, id 18625) (ttl 79, id 57860)

01:48:00 192.168.0.1.1063 > 172.16.0.1.80: . [tcp sum ok] 232:232(0) ack 306129 win 7312 <nop,nop,timestamp 32265 447226430> (DF) [tos 0xf (EC)] (ttl 116, id 49672)

01:48:00 172.16.0.1.80 > 192.168.0.1.1063: . 307577:309025(1448) ack 232 win 17376 <nop,nop,timestamp 447226431 32265> (DF) (ttl 64, id 27896)

01:48:00 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 187, id 44726) (ttl 132, id 30216)

01:48:00 192.168.0.1.1063 > 172.16.0.1.80: . [tcp sum ok] 232:232(0) ack 307577 win 8760 <nop,nop,timestamp 32266 447226430> (DF) [tos 0xf (EC)] (ttl 116, id 49928)

01:48:00 172.16.0.1.80 > 192.168.0.1.1063: . 309025:310473(1448) ack 232 win 17376 <nop,nop,timestamp 447226432 32266> (DF) (ttl 64, id 12684)

01:48:00 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 252, id 6804) (ttl 201, id 43029)

01:48:00 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 173, id 33976) (ttl 182, id 4152)

01:48:01 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 194, id 1202) (ttl 239, id 59037)

01:48:01 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 215, id 21023) (ttl 188, id 33475)

01:48:01 192.168.0.1 > 172.16.0.1: icmp: source quench for 172.16.0.1.80 > 192.168.0.1.1063: [|tcp] (ttl 108, id 18051) (ttl 109, id 56328)

Solucionando la vulnerabilidad

RTFS

(Leyendo las especificaciones)

No debería estar recibiéndolos....

El RFC 1122 dice:

“hosts MUST react to ICMP Source Quench messages by slowing transmission on the connection”

Sin embargo, el RFC 1812 dice:

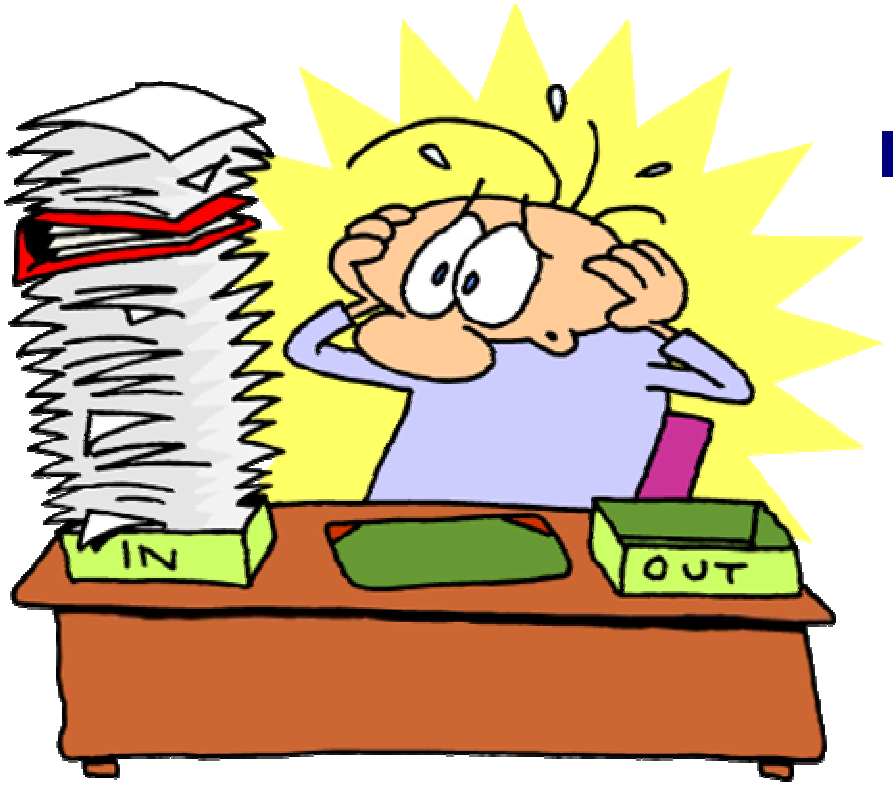
“A router SHOULD NOT originate ICMP Source Quench messages”

....entonces simplemente ignórelos

De acuerdo a las especificaciones de la IETF, los routers no deberían enviar mensajes ICMP Source Quench. Asimismo, TCP implementa su propio mecanismo de control de flujo, que no hace uso de los mensajes ICMP Source Quench. Por lo tanto,

Los hosts deberían **ignorar** aquellos mensajes ICMP Source Quench destinados a conexiones TCP

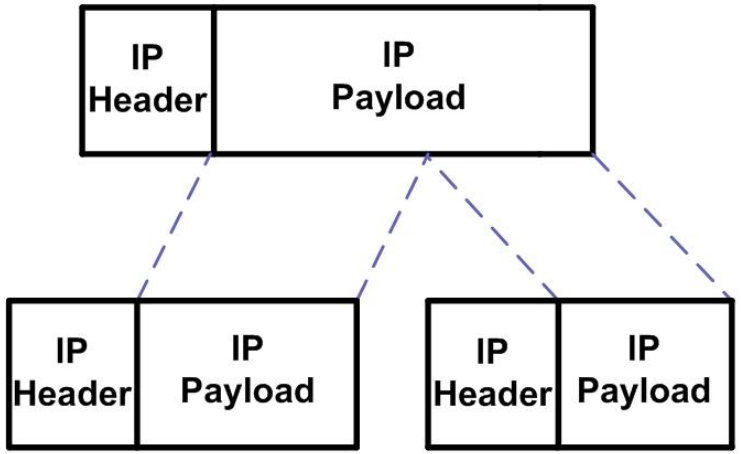
Blind performance-degrading attack



- Degradando la performance de una conexión TCP arbitraria

Fragmentación IP

- El protocolo IP brinda un mecanismo para poder “fragmentar” los paquetes.
- Cuando un sistema deba enviar un paquete IP a traves de un enlace cuyo MTU es menor que el tamaño del paquete, dicho sistema fragmentará el paquete IP en cuestión.
- Cada fragmento tendrá su propio encabezado IP.
- Todos los fragmentos tendrán en común el mismo **IP ID**.
- El **offset** de cada uno de los fragmentos será diferente.
- Todos los fragmentos, salvo el último, tendrán el bit **MF** (More Fragments) en 1.



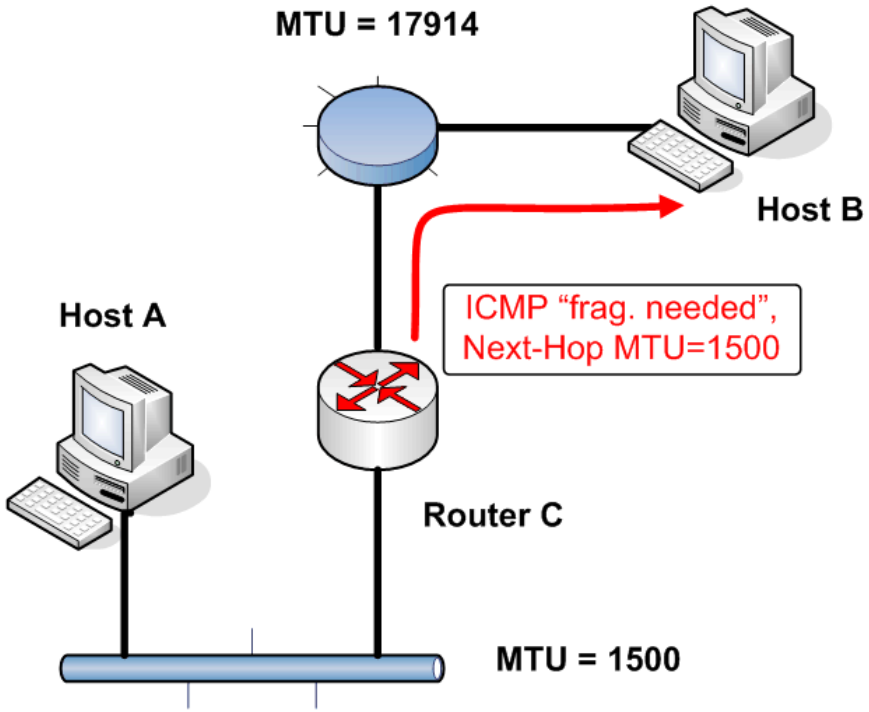
Algunos problemas de la fragmentación

- Si se pierde un solo fragmento, se debe desechar los demás.
- Una vez recibido un fragmento, es difícil estimar cuánto tiempo se debe “esperar” a que lleguen los demás.
- El campo IP ID es de solamente 16 bits. Con el ancho de banda disponible actualmente, es fácil que en un instante dado existan en la red fragmentos con un mismo IP ID, pero correspondientes a distintos paquetes. Es decir, es muy probable que un mismo IP ID se deba reutilizar antes que lo esperado.
- Esto último abre la puerta a un vector de ataque: un atacante podría falsificar fragmentos con cada uno de los distintos IP ID, para así confundir al sistema atacado. Dicho sistema no podría discernir entre los fragmentos legítimos, y de los que forman parte del ataque.

Mecanismo Path-MTU Discovery

- El mecanismo de fragmentación permite que sistemas con distintos MTUs puedan interoperar
- A fines de los años 80, un trabajo de investigación (“Fragmentation considered harmful”, Mogul) explicitó los problemas del mecanismo de fragmentación
- La IETF diseñó un mecanismo para evitar la necesidad de fragmentar paquetes, y lo llamó “Path-MTU Discovery” (ó “Descubrimiento de la Máxima Unidad de Transferencia del Camino”).

Operación del mecanismo Path-MTU Discovery

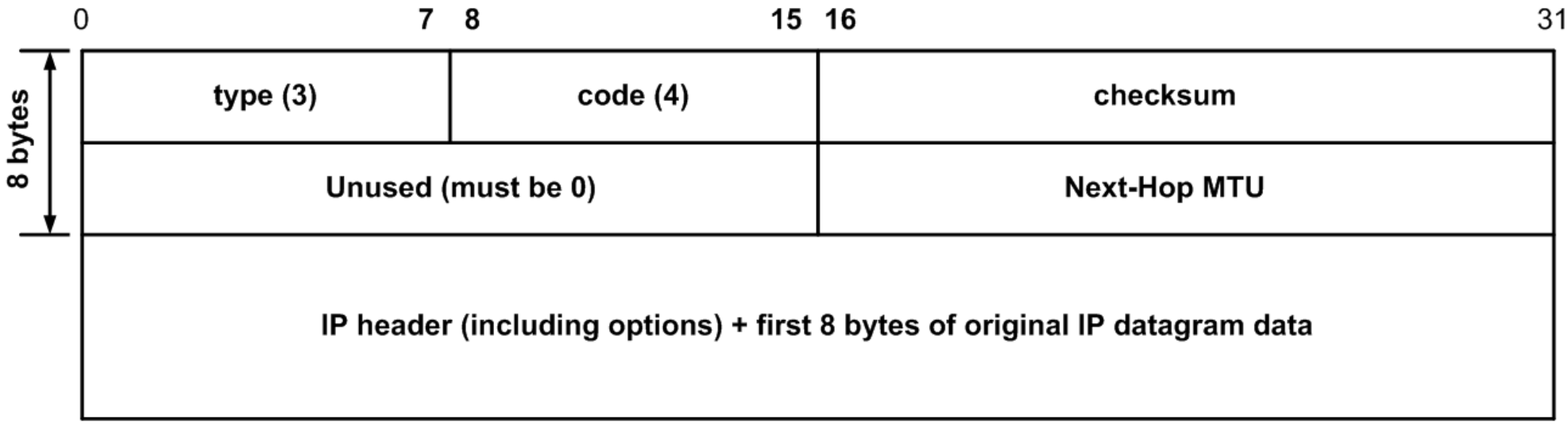


- Los paquetes IP son enviados con el bit DF (“don’t fragment”) seteado.
- Si algún router encuentra que no puede reenviar un paquete sin fragmentarlo, y detecta que el bit DF está seteado, descartará el paquete y enviará un mensaje de error ICMP “fragmentation needed and DF bit set” al host que envió el paquete en cuestión. Dicho mensaje de error contendrá el valor del MTU que no permite que el paquete se envíe sin ser fragmentado.

Operación del mecanismo Path-MTU Discovery *(continuado)*

- Al recibir este mensaje ICMP, el host en cuestión reducirá el tamaño de los paquetes que envía, consecuentemente.
- Este proceso se repetirá iterativamente hasta que los paquetes comiencen a llegar a destino.
- En tal punto, el sistema que se encuentra enviando información habrá descubierto el MTU del camino, evitándose así la necesidad de fragmentar paquetes.

Formato del paquete ICMP “fragmentation needed but DF bit set”



- De acuerdo a las especificaciones de la IETF, el valor contenido en el campo “Next-Hop MTU” puede ser tan chico como “68”. Dicho valor se conoce como el mínimo MTU de IPv4.

Blind performance-degrading attack

Un atacante podría falsificar un mensaje ICMP “fragmentation needed and DF bit set” que reporte un Next-Hop MTU bajo (tan bajo como 68 bytes).

Como resultado, el host atacado reduciría el tamaño de los paquetes que el mismo envía, al tamaño reportado por el campo Next-Hop MTU del mensaje de error ICMP. Solamente minutos mas tarde el tamaño de los paquetes podría volver a ser incrementado. Ási,

- El overhead (proporción headers/data) sería incrementado, llevando a una reducción en la tasa de transferencia.
- Para lograr mantener ela misma tasa de transferencia que antes del ataque, el sistema atacado debería incrementar el **packet rate,** lo que llevaría a una degradación de la performance general del sistema en cuestión.

Impacto del ataque

Un atacante podría reducir la performance de una conexión TCP arbitraria, incluso estando fuera del camino que siguen los paquetes pertenecientes a la conexión atacada

La tasa de transferencia se degradaría como consecuencia del aumento de overhead (proporción headers/data)

Para mantener la misma tasa de transferencia alcanzada previo al ataque, el **packet rate** debería ser incrementado, llevando esto a una degradación de la performance general de los sistemas involucrados

“Casos especiales”

(the height of irony)

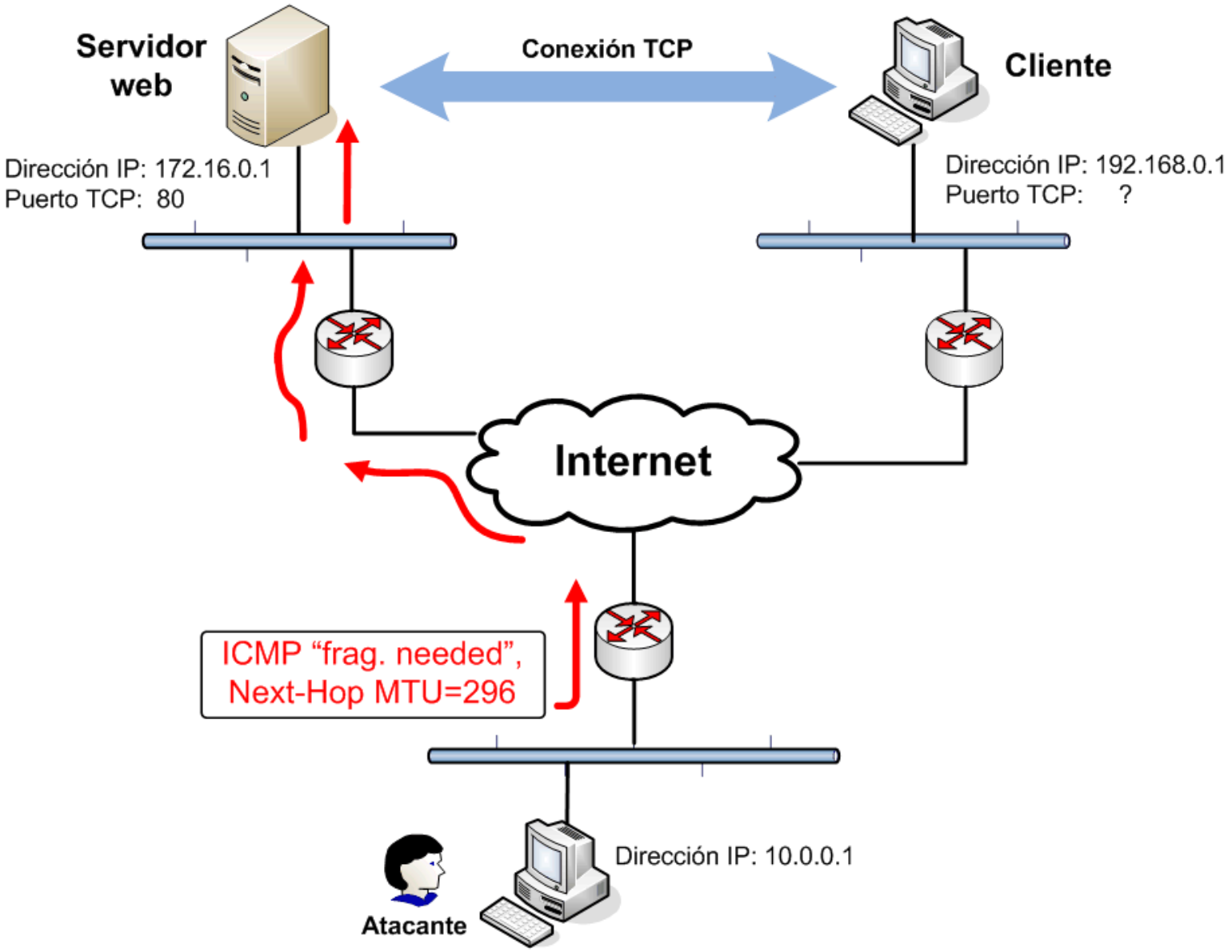
- En determinados escenarios, podría suceder que, debido al uso de opciones IP y/o de opciones TCP, se utilizaran mas de 68 bytes para encabezamientos.
- Un caso particular de este escenario sería aquél en el cual se protege una conexión TCP mediante IPSec.
- Si un atacante lograra reducir el Path-MTU asumido para la conexión a un tamaño menor o igual a la cantidad de bytes utilizada para los encabezamientos, el resultado del ataque sería “imprevisible”.
- Probablemente, la conexión TCP atacada se “congelaría”, el equipo se “colgaría”, o ambos.
- Es decir, utilizando IPSec estaríamos “disparándonos a nuestro propios pies”

Aplicaciones afectadas

Las aplicaciones más afectadas serán aquellas que dependen de conexiones de alta performance para un correcto funcionamiento.

Si el objetivo del ataque es BGP, este ataque podría llevar a inconsistencias en la información de ruteo, con la posibilidad de denegación de servicio a redes enteras

Posible escenario de un ataque



Salida de tcpdump

07:33:33.450711 192.168.0.1.80 > 10.0.0.1.3028: . 55381:56801(1420) ack 0 win 17040 (DF) (ttl 64, id 46716)
07:33:33.766220 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 48281 win 9940 (DF) (ttl 117, id 55756)
07:33:33.766241 192.168.0.1.80 > 10.0.0.1.3028: . 56801:58221(1420) ack 0 win 17040 (DF) (ttl 64, id 64806)
07:33:33.770295 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 49701 win 9940 (DF) (ttl 117, id 55758)
07:33:33.770318 192.168.0.1.80 > 10.0.0.1.3028: . 58221:59641(1420) ack 0 win 17040 (DF) (ttl 64, id 37411)
07:33:34.203906 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 51121 win 9940 (DF) (ttl 117, id 55760)
07:33:34.203962 192.168.0.1.80 > 10.0.0.1.3028: . 59641:61061(1420) ack 0 win 17040 (DF) (ttl 64, id 49486)
07:33:34.378908 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 52541 win 9940 (DF) (ttl 117, id 55761)
07:33:34.378933 192.168.0.1.80 > 10.0.0.1.3028: . 61061:62481(1420) ack 0 win 17040 (DF) (ttl 64, id 54369)
07:33:34.429296 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 53961 win 9940 (DF) (ttl 117, id 55762)
07:33:34.429318 192.168.0.1.80 > 10.0.0.1.3028: . 62481:63901(1420) ack 0 win 17040 (DF) (ttl 64, id 38867)
07:33:34.769519 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 55381 win 9940 (DF) (ttl 117, id 55764)
07:33:34.769565 192.168.0.1.80 > 10.0.0.1.3028: . 63901:65321(1420) ack 0 win 17040 (DF) (ttl 64, id 45655)
07:33:35.110200 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 56801 win 9940 (DF) (ttl 117, id 55765)
07:33:35.110259 192.168.0.1.80 > 10.0.0.1.3028: . 65321:66741(1420) ack 0 win 17040 (DF) (ttl 64, id 47463)
07:33:35.113291 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 58221 win 9940 (DF) (ttl 117, id 55766)
07:33:35.113314 192.168.0.1.80 > 10.0.0.1.3028: . 66741:68161(1420) ack 0 win 17040 (DF) (ttl 64, id 52075)
07:33:35.510370 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 59641 win 9940 (DF) (ttl 117, id 55769)
07:33:35.510393 192.168.0.1.80 > 10.0.0.1.3028: . 68161:69581(1420) ack 0 win 17040 (DF) (ttl 64, id 50555)
07:33:35.687634 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 61061 win 9940 (DF) (ttl 117, id 55770)
07:33:35.687656 192.168.0.1.80 > 10.0.0.1.3028: . 69581:71001(1420) ack 0 win 17040 (DF) (ttl 64, id 61555)
07:33:36.009372 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 62481 win 9940 (DF) (ttl 117, id 55771)
07:33:36.009398 192.168.0.1.80 > 10.0.0.1.3028: . 71001:72421(1420) ack 0 win 17040 (DF) (ttl 64, id 34296)
07:33:36.180815 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 63901 win 9940 (DF) (ttl 117, id 55773)
07:33:36.180873 192.168.0.1.80 > 10.0.0.1.3028: . 72421:73841(1420) ack 0 win 17040 (DF) (ttl 64, id 53575)
07:33:36.506718 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 65321 win 9940 (DF) (ttl 117, id 55774)

Salida de tcpdump *(cont.)*

```
07:33:36.506743 192.168.0.1.80 > 10.0.0.1.3028: . 73841:75261(1420) ack 0 win 17040 (DF) (ttl 64, id 34721)
07:33:36.510804 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 66741 win 9940 (DF) (ttl 117, id 55775)
07:33:36.510827 192.168.0.1.80 > 10.0.0.1.3028: . 75261:76681(1420) ack 0 win 17040 (DF) (ttl 64, id 34894)
07:33:37.004635 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 68161 win 9940 (DF) (ttl 117, id 55777)
07:33:37.004695 192.168.0.1.80 > 10.0.0.1.3028: . 76681:78101(1420) ack 0 win 17040 (DF) (ttl 64, id 44036)
07:33:37.008756 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 69581 win 9940 (DF) (ttl 117, id 55778)
07:33:37.008779 192.168.0.1.80 > 10.0.0.1.3028: . 78101:79521(1420) ack 0 win 17040 (DF) (ttl 64, id 47291)
07:33:37.213783 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 71001 win 9940 (DF) (ttl 117, id 55779)
07:33:37.213844 192.168.0.1.80 > 10.0.0.1.3028: . 79521:80941(1420) ack 0 win 17040 (DF) (ttl 64, id 37137)
07:33:37.222434 10.0.0.1 > 192.168.0.1: icmp: 10.0.0.1 unreachable - need to frag (mtu 296) (ttl 232, id 5693)
07:33:37.222472 192.168.0.1.80 > 10.0.0.1.3028: . 71001:71257(256) ack 0 win 17040 (DF) (ttl 64, id 58749)
07:33:37.222578 192.168.0.1.80 > 10.0.0.1.3028: . 71257:71513(256) ack 0 win 17040 (DF) (ttl 64, id 51033)
07:33:37.222845 192.168.0.1.80 > 10.0.0.1.3028: . 71513:71769(256) ack 0 win 17040 (DF) (ttl 64, id 32798)
07:33:37.223118 192.168.0.1.80 > 10.0.0.1.3028: . 71769:72025(256) ack 0 win 17040 (DF) (ttl 64, id 41766)
07:33:37.778893 10.0.0.1 > 192.168.0.1: icmp: 10.0.0.1 unreachable - need to frag (mtu 296) (ttl 221, id 2124)
07:33:37.778917 192.168.0.1.80 > 10.0.0.1.3028: . 71001:71257(256) ack 0 win 17040 (DF) (ttl 64, id 36548)
07:33:37.779022 192.168.0.1.80 > 10.0.0.1.3028: . 71257:71513(256) ack 0 win 17040 (DF) (ttl 64, id 61732)
07:33:37.779290 192.168.0.1.80 > 10.0.0.1.3028: . 71513:71769(256) ack 0 win 17040 (DF) (ttl 64, id 51438)
07:33:37.779563 192.168.0.1.80 > 10.0.0.1.3028: . 71769:72025(256) ack 0 win 17040 (DF) (ttl 64, id 40154)
07:33:37.804701 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 72421 win 9940 (DF) (ttl 117, id 55782)
07:33:37.804726 192.168.0.1.80 > 10.0.0.1.3028: . 72421:72677(256) ack 0 win 17040 (DF) (ttl 64, id 59091)
07:33:37.804829 192.168.0.1.80 > 10.0.0.1.3028: . 72677:72933(256) ack 0 win 17040 (DF) (ttl 64, id 53746)
07:33:37.805098 192.168.0.1.80 > 10.0.0.1.3028: . 72933:73189(256) ack 0 win 17040 (DF) (ttl 64, id 48719)
07:33:37.805371 192.168.0.1.80 > 10.0.0.1.3028: . 73189:73445(256) ack 0 win 17040 (DF) (ttl 64, id 37796)
07:33:38.000265 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 73841 win 9940 (DF) (ttl 117, id 55783)
07:33:38.000287 192.168.0.1.80 > 10.0.0.1.3028: . 73841:74097(256) ack 0 win 17040 (DF) (ttl 64, id 65040)
07:33:38.000391 192.168.0.1.80 > 10.0.0.1.3028: . 74097:74353(256) ack 0 win 17040 (DF) (ttl 64, id 50520)
```

Salida de tcpdump *(cont.)*

```
07:33:38.000660 192.168.0.1.80 > 10.0.0.1.3028: . 74353:74609(256) ack 0 win 17040 (DF) (ttl 64, id 37744)
07:33:38.000933 192.168.0.1.80 > 10.0.0.1.3028: . 74609:74865(256) ack 0 win 17040 (DF) (ttl 64, id 49194)
07:33:38.001205 192.168.0.1.80 > 10.0.0.1.3028: . 74865:75121(256) ack 0 win 17040 (DF) (ttl 64, id 35292)
07:33:38.001478 192.168.0.1.80 > 10.0.0.1.3028: . 75121:75377(256) ack 0 win 17040 (DF) (ttl 64, id 35405)
07:33:38.001751 192.168.0.1.80 > 10.0.0.1.3028: . 75377:75633(256) ack 0 win 17040 (DF) (ttl 64, id 50111)
07:33:38.002024 192.168.0.1.80 > 10.0.0.1.3028: . 75633:75889(256) ack 0 win 17040 (DF) (ttl 64, id 63637)
07:33:38.002297 192.168.0.1.80 > 10.0.0.1.3028: . 75889:76145(256) ack 0 win 17040 (DF) (ttl 64, id 37723)
07:33:38.099395 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 75261 win 9940 (DF) (ttl 117, id 55784)
07:33:38.099413 192.168.0.1.80 > 10.0.0.1.3028: . 76145:76401(256) ack 0 win 17040 (DF) (ttl 64, id 44236)
07:33:38.099518 192.168.0.1.80 > 10.0.0.1.3028: . 76401:76657(256) ack 0 win 17040 (DF) (ttl 64, id 58026)
07:33:38.099786 192.168.0.1.80 > 10.0.0.1.3028: . 76657:76913(256) ack 0 win 17040 (DF) (ttl 64, id 55844)
07:33:38.100058 192.168.0.1.80 > 10.0.0.1.3028: . 76913:77169(256) ack 0 win 17040 (DF) (ttl 64, id 53944)
07:33:38.295212 10.0.0.1.3028 > 192.168.0.1.80: . [tcp sum ok] ack 76681 win 9940 (DF) (ttl 117, id 55786)
07:33:38.295271 192.168.0.1.80 > 10.0.0.1.3028: . 77169:77425(256) ack 0 win 17040 (DF) (ttl 64, id 50067)
07:33:38.295377 192.168.0.1.80 > 10.0.0.1.3028: . 77425:77681(256) ack 0 win 17040 (DF) (ttl 64, id 50320)
07:33:38.295645 192.168.0.1.80 > 10.0.0.1.3028: . 77681:77937(256) ack 0 win 17040 (DF) (ttl 64, id 46302)
07:33:38.295918 192.168.0.1.80 > 10.0.0.1.3028: . 77937:78193(256) ack 0 win 17040 (DF) (ttl 64, id 35011)
07:33:38.296190 192.168.0.1.80 > 10.0.0.1.3028: . 78193:78449(256) ack 0 win 17040 (DF) (ttl 64, id 33938)
07:33:38.296463 192.168.0.1.80 > 10.0.0.1.3028: . 78449:78705(256) ack 0 win 17040 (DF) (ttl 64, id 34572)
07:33:38.296736 192.168.0.1.80 > 10.0.0.1.3028: . 78705:78961(256) ack 0 win 17040 (DF) (ttl 64, id 50867)
07:33:38.297009 192.168.0.1.80 > 10.0.0.1.3028: . 78961:79217(256) ack 0 win 17040 (DF) (ttl 64, id 49460)
07:33:38.297282 192.168.0.1.80 > 10.0.0.1.3028: . 79217:79473(256) ack 0 win 17040 (DF) (ttl 64, id 54046)
07:33:38.297554 192.168.0.1.80 > 10.0.0.1.3028: . [tcp sum ok] 79473:79497(24) ack 0 win 17040 (DF) (ttl 64, id
57102)
```

Analizando posibles soluciones a la vulnerabilidad

Alternativas posibles para enfrentar el problema

Potenciales soluciones

- Seguir utilizando el mecanismo de fragmentación IP
- Enviar paquetes pequeños
- Diseñar un mecanismo Path-MTU Discovery que no utilice ICMP en absoluto
- Mejorar el mecanismo Path-MTU Discovery actual

Utilizar el mecanismo de fragmentación, o enviar paquetes pequeños

- El mecanismo de fragmentación tiene muchos potenciales problemas. Los mismos fueron descriptos a fines de los años ’80 en el trabajo “Fragmentation considered harmful” de Jerry Mogul, y, mas recientemente en el trabajo “Fragmentation considered VERY harmful” de Heffner et al. Por otro lado, los routers IPv6 no fragmentan paquetes.
- En el caso de IPv4, el MTU mínimo asegurado por las especificaciones de la IETF es de 68 bytes, por lo cual “enviar paquetes pequeños” **es inaceptable**. En el caso de IPv6, el MTU mínimo es de 1280 bytes, por lo cual “enviar paquetes pequeños” **podría** ser una opción.

Diseñar un mecanismo PMTUD que no dependa de ICMP

- La IETF recientemente a estandarizado (RFC4821) un mecanismo que no depende de mensajes ICMP para descubrir el Path-MTU.
- El mismo se basa en timeouts para suponer la condición de “paquete muy grande”.
- Sin embargo, sin la ayuda de ICMP, el mecanismo resultante tiene un **tiempo de convergencia elevado**.
- Asimismo, en redes con una perdida de paquetes relativamente elevada, el mecanismo se podría volver extremadamente ineficiente.

Mejorando el mecanismo PMTUD actual

Análisis de posibles mejoras al mecanismo Path-MTU Discovery actual

Validando los mensajes ICMP

- Si se recibe un paquete ICMP “fragmentation needed and DF bit set”, el segmento TCP correspondiente debería haber sido descartado. Por ello, si el mensaje ICMP se refiere a un segmento sobre el cual ya se ha recibido un acuse de recibo, descartaremos el mensaje ICMP en cuestión.
- Los mensajes de error ICMP son causados por paquetes enviados. Si el mensaje ICMP recibido se refiere a datos que todavía no han sido enviados, entonces sería válido descartar el mensaje ICMP en cuestión.

TCP debería chequear que el TCP sequence number contenido en el cuerpo del mensaje ICMP esté en el rango:

SND.UNA =< SEG.SEQ < SND.NXT

Esto significa que los mensajes ICMP deberían corresponder a datos envíados, sobre los cuales todavía no se ha recibido un acuse de recibo

Propiedades de la validación realizada

- Para conexiones inactivas, se elimina la vulnerabilidad
- Para conexiones activas, el atacante tendrá una chance de TCP_Window/2^^32 . Simplemente estamos requiriendo mayor esfuerzo al atacante.
- Si bien la mitigación proporcionada parecería ser suficiente, la tendencia a utilizar ventanas TCP (TCP_Window) cada vez mas elevadas, y la tendencia a que grandes anchos de banda sean cada vez mas comunes, hacen este chequeo insuficiente.
  - En un mediano plazo nos estaríamos enfrentando nuevamente a la misma vulnerabilidad.
  - Lo único aceptable es solucionar la vulnerabilidad de una vez, y para siempre. (proactivamente)

Incrementando la protección (I)

- Si el mensaje de error ICMP recibido fuera legítimo, el segmento TCP al cual el mismo se refiere debería haber sido descartado.
- La única forma de saber si esto ha ocurrido es esperar un cierto tiempo, para ver si se recibe (o no) un acuse de recibo de los datos en cuestión.
- Así, resultaría lógico esperar al menos un RTO antes de actuar sobre el mensaje ICMP recibido, para darle tiempo a nuestros paquetes a que lleguen a destino, y a los correspondientes acuses de recibo a que lleguen a nuestro sistema

Incrementando la protección (II)

- Requerir que TCP espere un RTO antes de procesar los mensajes ICMP en cuestión podría llevar a tiempos de convergencia muy elevados.
- En particular, podría haber un gran delay antes de que los paquetes comiencen a llegar a destino.
- Las aplicaciones interactivas sufren considerablemente los tiempos de respuesta elevados.
- Sin embargo, es aceptable tener tiempos de respuesta levemente mas elevados **durante** la vida de una conexión, debido a cambios en PMTU dados por cambios en el camino que siguen los paquetes para llegar a destino.

Solución a la vulnerabilidad del mecanismo Path-MTU Discovery

Obteniendo una solución de ingeniería que incremente la seguridad del mecanismo, pero mantenga un buen tiempo de convergencia

Nuestra solución de ingeniería

- Validar los mensajes ICMP para asegurarse que los mismos fueron causados por datos envíados, de los cuales todavía no se ha acusado el recibo. Es decir, chequear el número de secuencia TCP contenido en el payload del mensaje ICMP recibido (`SND.UNA =< SEG.SEQ < SND.NXT`).
- Dividir el PMTUD en dos fases: Initial PMTUD, y PMTU Update
- En la fase Initial PMTUD, los mensajes ICMP “Packet Too Big” se procesan tan pronto como son recibidos
- En la fase PMTU Update, se espera al menos un RTO antes de que los mismos sean procesados
- De este modo, podemos lograr para conexiones nuevas el mismo tiempo de convergencia que el mecanismo PMTUD tradicional (y así, no afectaremos a las aplicaciones interactivas), pero seremos resistentes al ataque.

Initial PMTUD y PMTU Update

- La fase Initial Path-MTU Discovery es cuando TCP trata de enviar segmentos que son más grandes que los que hasta el momento han sido enviados por este sistema, y recibidos por el sistema remoto, para esta conexión. Durante esta fase, TCP no tiene conocimiento previo de que segmentos de este tamaño puedan llegar a destino, y por lo tanto es adecuado confiar en “la red” cuando la misma nos reporta un error.
- La fase Path-MTU Update es cuando TCP trata de enviar segmentos que son menores o iguales que los que ya se han transferido con éxito al sistema remoto. Durante esta fase, TCP si tiene conocimiento que segmentos de este tamaño pueden llegar a destino sin ser fragmentados. Por ellos, debemos ser mas cuidadosos con los errores reportados por la red.
- Para poder diferenciar entre las dos fases, simplemente debemos recordar cual es el maximo tamaño de segmento transmitido hasta el momento (`maxsizesent`), y cual es el máximo tamaño se segmento recibido por el extremo remoto hasta el momento (`maxsizeacked`), y compararlo con el Next-Hop MTU reportado por el mensaje de error ICMP recibido (`claimedmtu`).

Initial PMTUD y PMTU Update

- Si claimedmtu > maxsizesent, entonces el mensaje de error tiene que haber sido falsificado. No pudo ser causado por algo que no enviamos!
- Si (claimedmtu < maxsizesent) && (claimedmtu > maxsizeacked), entonces estamos en la fase Initial Path-MTU Discovery.
- Si (claimedmtu > maxsizesent) && (claimedmtu < maxsizeacked), estaremos en la fase Path-MTU Update

Procesamiento de mensajes ICMP

- En la fase Initial PMTUD, procesaremos los mensajes ICMP tan pronto como sean recibidos (PMTUD tradicional).
- En la fase PMTU Update, simplemente “recordaremos” el mensaje recibido, y señalizaremos una condición de “error pendiente”
- Cada vez que se reciba un acuse de recibo, si existe “un error pendiente”, nos fijaremos si dicho ACK acusa el recibo del TCP SEQ que estaba incluido en el payload del mensaje ICMP recibido. Si lo hace, deberemos descartar el mensaje ICMP en cuestión, y “limpiar” la condición de “error pendiente”.
- Cada vez que un segmento expire, nos fijaremos si existía un “error pendiente” con referencia al mismo. Si así lo fuera, en ese momento procesaremos el mensaje ICMP de error en cuestión.

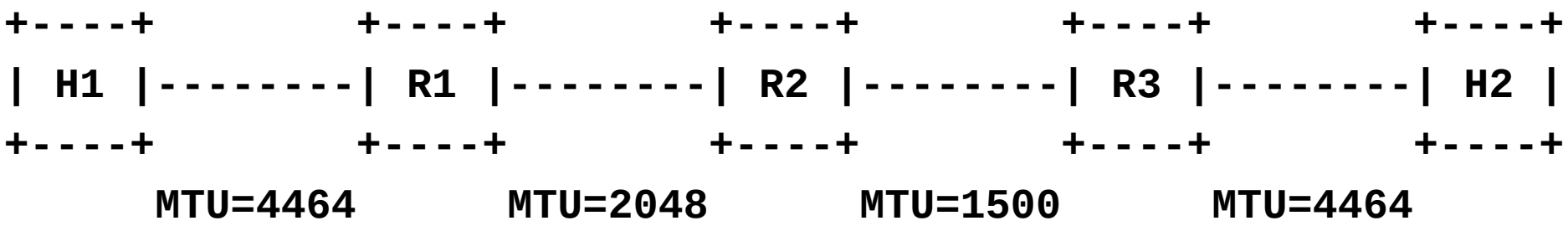
Consideraciones de seguridad de la solución propuesta

- Las conexiones inactivas no son vulnerables, ya que para ellas **`SND.UNA == SND.NXT`**
- La fase Initial PMTUD es realmente corta como para preocuparnos
- Para conexiones en la fase PMTU Update, el atacante no sólo debería poder adivinar el TCP SEQ correcto, sino que también debería ser capaz de eliminar paquetes pertenecientes a la conexión. Si esto fuera posible, el atacante ya habría logrado un DoS!

La contramedida en acción

Escenarios posibles

Escenario hipotético

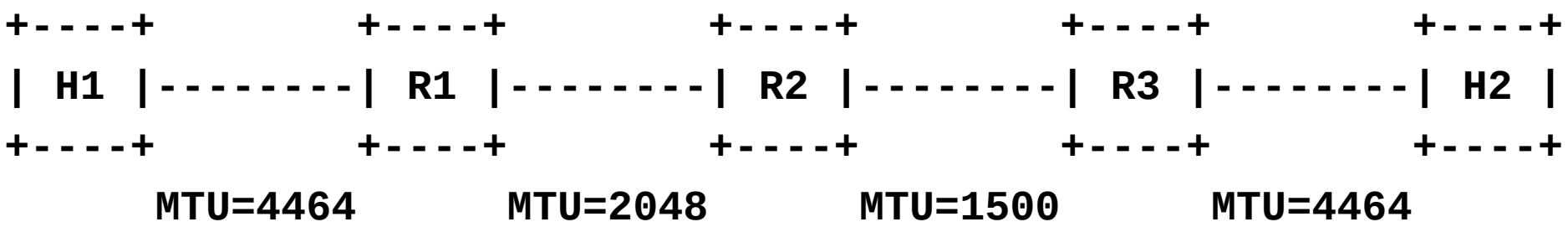


- Dos hosts se encuentran “conectados” a través de tres routers
- El host H1 establecerá una conexión con el host H2, y transferirá información al mismo

Escenario 1: Operación normal en transferencias “bulk”

```
   Host 1                                                Host 2

1.    -->            <SEQ=100><CTL=SYN>                -->
2.    <--      <SEQ=X><ACK=101><CTL=SYN,ACK>           <--
3.    -->       <SEQ=101><ACK=X+1><CTL=ACK>            -->
4.    --> <SEQ=101><ACK=X+1><CTL=ACK><DATA=4424>       -->
5.       <--- ICMP "Packet Too Big" MTU=2048, TCPseq#=101 <--- R1
6.    --> <SEQ=101><ACK=X+1><CTL=ACK><DATA=2008>       -->
7.       <--- ICMP "Packet Too Big" MTU=1500, TCPseq#=101 <--- R2
8.    --> <SEQ=101><ACK=X+1><CTL=ACK><DATA=1460>       -->
9.    <--       <SEQ=X+1><ACK=1561><CTL=ACK>           <--
```



Escenario 2: Operación durante cambios de Path-MTU

```
   Host 1                                                Host 2

1.                    (El Path-MTU se reduce)
2.     -->  <SEQ=100><ACK=X><CTL=ACK><DATA=1500>   -->
3.        <--- ICMP "Packet Too Big" MTU=1492, TCPseq#=100 <--- R2
4.                     (El segmento expira)
5.     -->  <SEQ=100><ACK=X><CTL=ACK><DATA=1452>   -->
6.     <--       <SEQ=X><ACK=1552><CTL=ACK>         <--
```

Escenario 3: Conexión inactiva siendo atacada

```
 Host 1                                                  Host 2

1.      -->      <SEQ=100><ACK=X><CTL=ACK><DATA=50>       -->
2.      <--          <SEQ=X><ACK=150><CTL=ACK>            <--
3.           <--- ICMP "Packet Too Big" MTU=68, TCPseq#=100 <---
4.           <--- ICMP "Packet Too Big" MTU=68, TCPseq#=100 <---
5.           <--- ICMP "Packet Too Big" MTU=68, TCPseq#=100 <---
```

Escenario 4: Conexión activa siendo atacada

```
 Host 1                                                  Host 2

1.     -->     <SEQ=100><ACK=X><CTL=ACK><DATA=1460>     -->
2.     -->     <SEQ=1560><ACK=X><CTL=ACK><DATA=1460>    -->
3.     -->     <SEQ=3020><ACK=X><CTL=ACK><DATA=1460>    -->
4.     -->     <SEQ=4480><ACK=X><CTL=ACK><DATA=1460>    -->
5.          <--- ICMP "Packet Too Big" MTU=68, TCPseq#=100 <---
6.     <--          <SEQ=X><CTL=ACK><ACK=1560>           <--
```

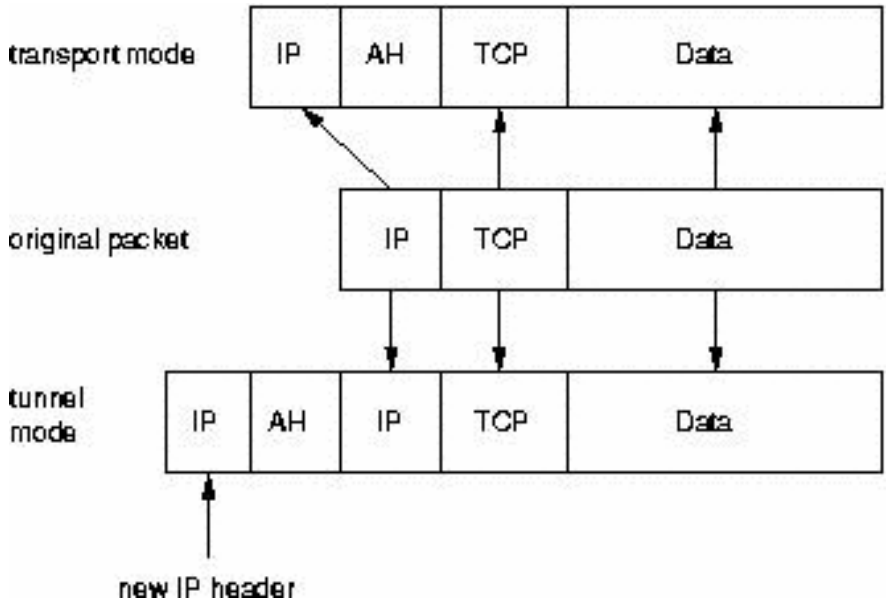
Ineficacia de los mecanismos existentes para mitigar ataques ICMP contra TCP

Porqué los mecanismos existentes son inefectivos contra los ataques ICMP

Ineficacia de otros mecanismos

- Ineficacia de IPSec
- Ineficacia de la opción TCP MD5
- Ineficacia de ingress-filtering y egress-filtering clásicos
- SSL

Autentificación en IPSec

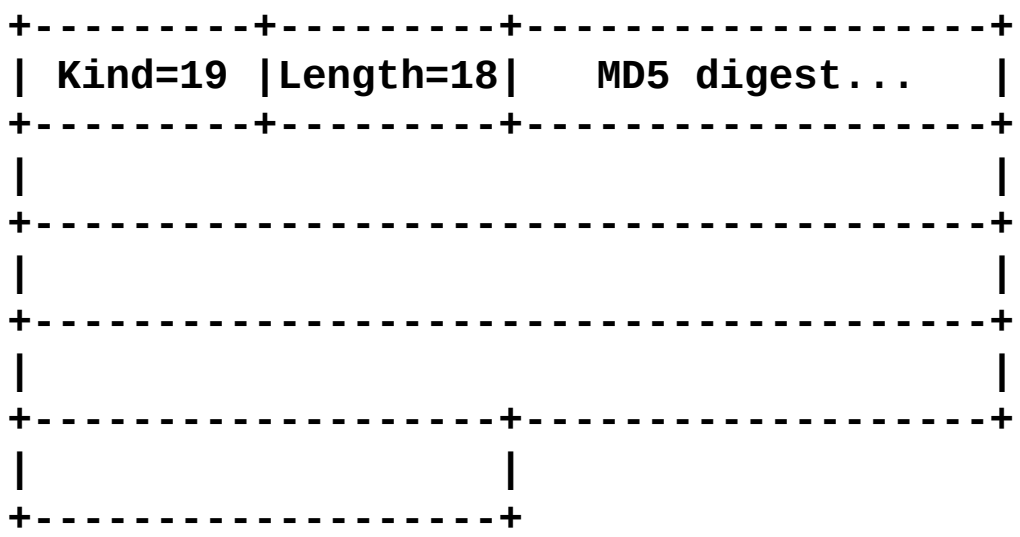


- El modo “transporte” sirve para proteger el IP payload
- El modo tunnel protege el paquete IP completo. Se utiliza principalmente para VPNs.
- Para utilizar los servicios de IPsec, se debe establecer primero una Asociación de Seguridad (SA).

Dificultades de IPsec para protegernos de estos ataques

- El bajo nivel de uso de protocolos para establecimiento dinámico de Asociaciones de Seguridad hace que IPSec no pueda ser utilizado en la práctica como contramedida para estos ataques.
- Asimismo, debido a que la especificación de ICMP no exige que se incluya en el IP payload el paquete original completo, no se podrá recalular la firma de los paquetes.
- Pese a que IPsec permite proteger el IP payload o el paquete IP completo, la norma no hace recomendaciones en lo referente a los mensajes ICMP. Simplemente sugiere que se pueda elegir entre filtrarlos y no-filtrarlos, así como también que en el caso de los “frag needed” se pueda establecer un límite inferior.

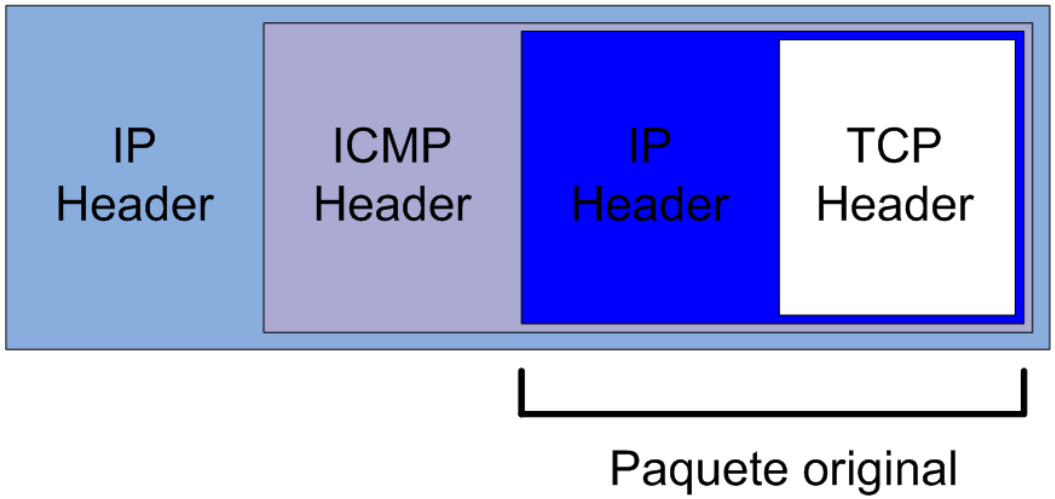
Opción TCP MD5



- La opción TCP MD5 provee autentificación del segmento TCP en cuestión.
- La especificación de la opción TCP MD5 ni siquiera menciona a los mensajes de error ICMP.
- La información contenida en el paquete ICMP no será suficiente como para recalcular la firma MD5

Ingress-filtering y egress-filtering

- La dirección IP origen de los mensajes ICMP no necesita ser falsificada.
- Así, la aplicación de ingress-filtering y egress-filtering clásico no servirá como contra-medida para estos ataques.
- Si bien la dirección IP origen contenida el el **payload** del mensaje ICMP sí necesita ser falsificada, pocos sistemas implementan este “filtrado mas avanzado”.



Ineficacia de SSL

- SSL protege la **información** transferida mediante una conexión TCP
- Su correcta operación depende de la correcta operación de TCP.
- En consecuencia, el uso de SSL no protege a las conexiones TCP de ninguno de los ataques descriptos en esta presentación.

Implementaciones abiertas

Quién implementa qué

Running code

(Implementación de las contra-medidas propuestas)

Implemented counter-measures	Connection reset	Throughput reduction	performance degrading
Linux	**Sí**	**Sí**	**Pendiente**
FreeBSD	**Sí**	**Sí**	**Pendiente**
NetBSD	**Sí**	**Sí**	**Sí**
OpenBSD	**Sí**	**Sí**	**Sí**
Solaris	**Sí**	**Sí**	**Parcialmente**

- Todas las implementaciones TCP/IP derivadas de BSD ó Mentat han implementado tradicionalmente el procesamiento de los llamados “errores graves” tal como se propone en este internet-draft por mas de 15 años.
- La mayoría de las implementaciones removieron el soporte de ICMP Source Quench para TCP debido al “disclosure” de NISCC.
- Virtualmente todas las implementaciones chequean ahora el TCP SEQ contenido en el ICMP payload.

Estado actual del internet-draft

- Las soluciones descriptas en esta presentación fueron propuestas en el ámbito de la IETF (Internet Engineering Task Force).
- La propuesta fue adoptada como elemento de trabajo del TCPM WG, para la categoría Informational (draft-ietf-tcpm-icmp-attacks).
- En consecuencia, las especificaciones de los protocolos **no** serán modificadas.

Preguntas?

Agradecimientos

- Leonardo Vidal & Carlos M. Martinez @ ANTEL
- ANTEL

Información de contacto

Fernando Gont

fernando@gont.com.ar

Más información en:

http://www.gont.com.ar