Sistemas Empotrados 2

Implementación de un Echo Server en XDP con una BBB

Nombre: Javier Julve Yubero NIP: 840710

Nombre: Javier Falcó Díez NIP: 797613

Introducción

Vamos a usar XDP_TX para implementar un servidor de eco ICMP/ICMPv6 en una Beagle Bone Black (BBB).

El servidor recibirá un paquete ICMP o ICMPv6, intercambiará las direcciones MAC de destino y origen, las direcciones IP/IPv6, cambiará el campo Type para que sea Response y cambiará el checksum para que sea correcto después de estas modificaciones.

Instalación debian y patch rt

Descargamos Debian 11 para la BBB

Instalamos patch rt mediante el comando:

```
sudo apt install bbb.io-kernel-5.10-ti-rt-am335x
```

Reiniciamos el sistema para que los cambios surtan efecto.

Configuración de red (uso de usb para red)

Para conectar la BBB a internet y al portátil hemos usado el cable de USB, no hemos usado los cables Ethernet. Hemos configurado el portátil para que haga forwarding de paquetes y usando iptables hemos puesto reglas para que funcione como un router para la BBB. Hemos configurado la BBB para que usara el portátil como gateway y le hemos agregado una dirección para uso de DNS. Esto ha servido para facilitar instalar utilidades en la BBB cómodamente, trabajar con git y comprobar que teníamos conexión con el exterior. Para IPv6 le hemos dado una dirección global a la BBB y la hemos configurado para que la salida por defecto sea la interfaz conectada por USB. Todas estas configuraciones están recogidas en dos scripts (disponibles en el repositorio), puesto que son volátiles y se eliminan al apagar los sistemas.

Explicación del funcionamiento del código

IPv4: Primero nos aseguramos que sea un paquete no vacío ni bugeado al comprobar que su inicio + 1 no es mayor al final del paquete, posteriormente gracias a las variables de xdp cambiamos las direcciones MAC en la capa de Ethernet. Comprobamos de nuevo lo mismo que al inicio para poder proseguir con la capa IP donde veremos si es un paquete ICMP, en caso afirmativo se le asignará un nuevo ttl y este dato junto al ttl anterior lo usaremos para poder calcular el checksum, el cual almacenaremos en otra variable de XDP y solo entonces haremos un PASS para poder devolver exitosamente nuestro reply creado en XDP.

IPv6: Hacemos básicamente lo mismo pero con una serie de datos establecidos por nosotros en forma de estructura privada, esto con el fin de definir una estructura ICMPv6 sin entrar en conflicto con las macros del sistema. El proceso en este caso es el mismo pero usando esta estructura, posteriormente en el caso de que sea un paquete ICMPv6, calculamos el checksum con diferencias de TTL y restando 23 (que es 17 en hexadecimal) con el fin de hacer que el checksum sea correcto. Ya que en nuestras pruebas tenía ese desfase.

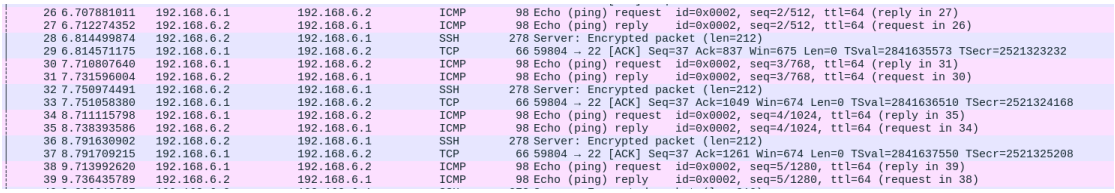
Explicación de las pruebas (tcpdump, wireshark)

Para probar el funcionamiento durante el desarrollo y con el programa final se han utilizado las herramientas de captura de paquetes tcpdump y Wireshark (que trabaja sobre tcpdump). tcpdump se ejecuta en el espacio de usuario y utiliza libpcap (Packet Capture Library), que es una biblioteca de espacio de usuario que usa PF_PACKET. PF_PACKET es un tipo de familia de socket en Linux que permite interactuar directamente con las tramas de red en la capa de enlace de datos.. Permite capturar paquetes directamente desde una interfaz de red antes de que sean procesados por las capas superiores del stack de red. Como XDP funciona por debajo de la pila de red del sistema operativo, si se captura con tcpdump en un equipo con un filtro XDP, los paquetes filtrados por XDP no aparecerán en la captura de tcpdump. De esta forma, si en un equipo (BBB) se carga el servidor eco y se captura con tcpdump y en otro equipo (portátil) se captura con tcpdump y se hace un ping al primero, en el primer equipo no se capturar´a ningún paquete ICMP y en el segundo se verá el tráfico de solicitudes y respuestas como si no hubiese ningún programa XDP. Así es fácil reconocer errores de en la forma de parseo y filtrado con XDP, porque los paquetes mal filtrados serán capturados por tcpdump en la BBB, y errores de manipulación de direcciones, TTL, checksum, etc porque serán capturados por el portátil. Utilizamos Wireshark en el portátil porque trabaja sobre tcpdump y tiene una interfaz más amigable.

```
sudo tcpdump -i nombre_interfaz icmp
```

Imágenes

Wireshark sin xdp



26	6.707881011	192.168.6.1	192.168.6.2	ICMP	98	Echo (ping) request id=0x0002, seq=2/512, ttl=64 (reply in 27)
27	6.712274352	192.168.6.2	192.168.6.1	ICMP	98	Echo (ping) reply id=0x0002, seq=2/512, ttl=64 (request in 26)
28	6.814499874	192.168.6.2	192.168.6.1	SSH	278	Server: Encrypted packet (len=212)
29	6.814571175	192.168.6.1	192.168.6.2	TCP	66	59804 → 22 [ACK] Seq=37 Ack=837 Win=675 Len=0 TSval=2841635573 TSecr=2521323232
30	7.710807640	192.168.6.1	192.168.6.2	ICMP	98	Echo (ping) request id=0x0002, seq=3/768, ttl=64 (reply in 31)
31	7.731596004	192.168.6.2	192.168.6.1	ICMP	98	Echo (ping) reply id=0x0002, seq=3/768, ttl=64 (request in 30)
32	7.750974491	192.168.6.2	192.168.6.1	SSH	278	Server: Encrypted packet (len=212)
33	7.751058380	192.168.6.1	192.168.6.2	TCP	66	59804 → 22 [ACK] Seq=37 Ack=1049 Win=674 Len=0 TSval=2841636510 TSecr=2521324168
34	8.711115798	192.168.6.1	192.168.6.2	ICMP	98	Echo (ping) request id=0x0002, seq=4/1024, ttl=64 (reply in 35)
35	8.738393586	192.168.6.2	192.168.6.1	ICMP	98	Echo (ping) reply id=0x0002, seq=4/1024, ttl=64 (request in 34)
36	8.791630902	192.168.6.2	192.168.6.1	SSH	278	Server: Encrypted packet (len=212)
37	8.791709215	192.168.6.1	192.168.6.2	TCP	66	59804 → 22 [ACK] Seq=37 Ack=1261 Win=674 Len=0 TSval=2841637550 TSecr=2521325208
38	9.713992620	192.168.6.1	192.168.6.2	ICMP	98	Echo (ping) request id=0x0002, seq=5/1280, ttl=64 (reply in 39)
39	9.736435789	192.168.6.2	192.168.6.1	ICMP	98	Echo (ping) reply id=0x0002, seq=5/1280, ttl=64 (request in 38)

Tcpdump con xdp

```
debian@BeagleBone:~/XDP-ICMP-Echo-Request-Handler$ sudo tcpdump -i usb1 icmp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on usb1, link-type EN10MB (Ethernet), snapshot length 262144 bytes
20:10:35.372024 IP 192.168.6.1 > 192.168.6.2: ICMP echo request, id 2, seq 1, length 64
20:10:35.372294 IP 192.168.6.2 > 192.168.6.1: ICMP echo reply, id 2, seq 1, length 64
20:10:36.371762 IP 192.168.6.1 > 192.168.6.2: ICMP echo request, id 2, seq 2, length 64
20:10:36.372046 IP 192.168.6.2 > 192.168.6.1: ICMP echo reply, id 2, seq 2, length 64
20:10:37.390488 IP 192.168.6.1 > 192.168.6.2: ICMP echo request, id 2, seq 3, length 64
20:10:37.390761 IP 192.168.6.2 > 192.168.6.1: ICMP echo reply, id 2, seq 3, length 64
20:10:38.396953 IP 192.168.6.1 > 192.168.6.2: ICMP echo request, id 2, seq 4, length 64
20:10:38.397224 IP 192.168.6.2 > 192.168.6.1: ICMP echo reply, id 2, seq 4, length 64
20:10:39.394442 IP 192.168.6.1 > 192.168.6.2: ICMP echo request, id 2, seq 5, length 64
20:10:39.394715 IP 192.168.6.2 > 192.168.6.1: ICMP echo reply, id 2, seq 5, length 64
```

Wireshark con xdp

	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.6.1	192.168.6.2	ICMP	98	Echo (ping) request id=0x0009, seq=1/256, ttl=64 (reply in 2)
2	0.003541719	192.168.6.2	192.168.6.1	ICMP	98	Echo (ping) reply id=0x0009, seq=1/256, ttl=64 (request in 1)
3	0.018757981	192.168.6.2	8.8.8.8	DNS	84	Standard query 0x5d82 PTR 1.6.168.192.in-addr.arpa
4	0.540113691	192.168.6.2	8.8.8.8	DNS	81	Standard query 0x5da2 A 1.debian.pool.ntp.org
5	0.540114168	192.168.6.2	8.8.8.8	DNS	81	Standard query 0x23ce AAAA 1.debian.pool.ntp.org
6	1.002586922	192.168.6.1	192.168.6.2	ICMP	98	Echo (ping) request id=0x0009, seq=2/512, ttl=64 (reply in 7)
7	1.015397705	192.168.6.2	192.168.6.1	ICMP	98	Echo (ping) reply id=0x0009, seq=2/512, ttl=64 (request in 6)
8	2.004651123	192.168.6.1	192.168.6.2	ICMP	98	Echo (ping) request id=0x0009, seq=3/768, ttl=64 (reply in 9)
9	2.021007907	192.168.6.2	192.168.6.1	ICMP	98	Echo (ping) reply id=0x0009, seq=3/768, ttl=64 (request in 8)
10	3.009764500	192.168.6.1	192.168.6.2	ICMP	98	Echo (ping) request id=0x0009, seq=4/1024, ttl=64 (reply in 11)
11	3.041844386	192.168.6.2	192.168.6.1	ICMP	98	Echo (ping) reply id=0x0009, seq=4/1024, ttl=64 (request in 10)
12	4.012647149	192.168.6.1	192.168.6.2	ICMP	98	Echo (ping) request id=0x0009, seq=5/1280, ttl=64 (reply in 13)
13	4.033103922	192.168.6.2	192.168.6.1	ICMP	98	Echo (ping) reply id=0x0009, seq=5/1280, ttl=64 (request in 12)
14	5.024413700	192.168.6.2	8.8.8.8	DNS	84	Standard query 0x5d82 PTR 1.6.168.192.in-addr.arpa
15	5.546022437	192.168.6.2	8.8.8.8	DNS	81	Standard query 0xa8b3 A 1.debian.pool.ntp.org
16	5.546022796	192.168.6.2	8.8.8.8	DNS	81	Standard query 0x6e16 AAAA 1.debian.pool.ntp.org

Tcpdump con xdp

```
debian@BeagleBone:~/XDP-ICMP-Echo-Request-Handler$ sudo tcpdump -i usb1 icmp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on usb1, link-type EN10MB (Ethernet), snapshot length 262144 bytes
```

Conclusión

Trabajar con este proyecto nos ha permitido ver cómo XDP puede ser una herramienta poderosa para personalizar el comportamiento de la red en el kernel, y cómo hacerlo requiere un profundo entendimiento de las capas de red y las estructuras de datos en C. Esto nos ha enseñado cómo podría aplicarse XDP en otros contextos, como la seguridad o el monitoreo de redes.

README

XDP ICMP Echo Request Handler

Este programa utiliza XDP (eBPF) para interceptar y modificar paquetes de red ICMP y ICMPv6. Su objetivo es interceptar solicitudes de eco ICMP (ping) y

ICMPv6, intercambiando las direcciones de origen y destino, y modificando el tipo del mensaje a la respuesta de eco correspondiente.

Funcionalidad

- **Intercambio de direcciones MAC**: Cambia las direcciones MAC de origen y destino en los paquetes Ethernet.
- **ICMP sobre IPv4**: Si el paquete es una solicitud de eco ICMP (Echo Request), intercambia las direcciones IP de origen y destino, cambia el tipo a Echo Reply y actualiza el checksum del paquete.
- **ICMP sobre IPv6**: Si el paquete es una solicitud de eco ICMPv6 (Echo Request), intercambia las direcciones IP de origen y destino, cambia el tipo a Echo Reply y actualiza el checksum del paquete.

¿Cómo funciona?

1. **XDP**: El programa se carga y ejecuta en el contexto de un XDP program, que es un tipo de programa eBPF que se ejecuta en la capa más baja de la pila de red, directamente en la interfaz de red.
2. **Procesamiento de paquetes**:

- Cuando un paquete ICMPv4 o ICMPv6 llega a la interfaz de red, el programa:
  1. Verifica si el paquete es ICMPv4 o ICMPv6.
  2. Si es una solicitud de eco (Echo Request), intercambia las direcciones de origen y destino.
  3. Modifica el tipo de ICMP a Echo Reply (respuesta de eco).
  4. Actualiza el checksum de ICMP (tanto para IPv4 como para IPv6) para reflejar el cambio de tipo de mensaje.

3. **Respuesta al paquete**: El programa luego reenvía el paquete con los cambios realizados.

Requisitos

- **Kernel**: 4.8 o superior.
- **Herramientas**: `clang`, `xdp`.

Dependencias

```
sudo apt install clang llvm libelf-dev libbpf-dev libpcap-dev build-essential
sudo apt install linux-headers-$(uname -r)
```

Debian:

```
sudo apt install linux-perf
```

Ubuntu:

```
sudo apt install linux-tools-$(uname -r)
```

Problemas

/usr/include/linux/types.h:5:10: fatal error: ‘asm/types.h’ file not found **

Encuentra donde está el archivo .h, podría no estar en /usr/include

```
find /usr/include/ -name types.h | grep asm
```

Haz un softlink entre los headers y donde la librería espera que estén:

```
sudo ln -s /usr/include/x86_64-linux-gnu/asm /usr/include/asm
```

Este problema puede ocurrir con otros headers requeridos dependiendo de la distribución de linux. Las soluciones proporcionadas arriba funcionan con cualquier otro header.

Uso

1. **Compilación**:

```
clang -O2 -target bpf -c echo_server.c -o echo_server.o
```

2. **Carga del programa**:

```
sudo ip link set dev <interface> xdp obj echo_server.o sec xdp
```

3. **Descarga del programa**:

```
sudo ip link set dev <interface> xdp off
```