net-firewall-simulator

Introduccion

El proyecto consiste en un **EDSL** cuyo objetivo es simular el comportamiento de un firewall de red. Dado un escenario predefinido (red de dispositivos y paquetes que atraviesan el firewall), el lenguaje permite replicar su funcionamiento. Está inspirado en *iptables* de UNIX, y las reglas que permite definir el firewall se asemejan a las de la tabla **filter** de este.

Archivos del simulador

Un archivo del simulador (extensión `.fws`) se divide en tres secciones. A continuación se detalla cada una de ellas junto con su sintaxis:

Sección network

Define los dispositivos de la red que serán protegidos por el firewall. Asi se definirían **N** dispositivos en la red:

```
network {
device device-name-1 {
mac = "???";
ip = ???.???.???.???;
subnet = ???.???.???.???/??;
interfaces = "???", "???", ..., "???";
}

device device-name-2 {
    mac = "???";
    ip = ???.???.???.???;
    subnet = ???.???.???.???/??;
    interfaces = "???", "???", ..., "???";
}
.
.
.
device device-name-N {
    mac = "???";
    ip = ???.???.???.???;
    subnet = ???.???.???.???/??;
    interfaces = "???", "???", ..., "???";
}

}
```

Cada dispositivo se describe con:

`mac:` debe ir encerrada entre `""`, se trata de una cadena de caracteres identificatoria.

`ip:` una dirección `IPv4` válida.

`subnet:` el rango de direcciones `IPv4` de la subred a la cual pertenece el dispositivo. Usar notación **CIDR** (más abajo hay un ejemplo). La subnet debe ser consistente con la dirección `IPv4` suministrada.

`interfaces:` una lista de cadenas de caracteres (encerrar entre `""`) que identifiquen a cada una de las interfaces disponibles del dispositivo, separadas por '`,`'.

> **IMPORTANT**
>
> Para que el programa funcione correctamente, se DEBE definir un dispositivo asociado al firewall, de nombre `firewall`. Por convención, consideramos al firewall como un dispositivo más en la red.

> **NOTE**
>
> Para asignar la subred del firewall, usamos el rango en el cual está contenida la ip publica del mismo. Por ejemplo, para un firewall cuya ip publica (para acceder a internet a traves del router) es 200.3.1.2, se podria usar un rango: 200.3.1.0/24. No afecta a los resultados de la simulacion, asi que es solo una convención.

Sección packets

Define los envíos que se llevarán a cabo durante la simulación, es decir, ¿qué máquina le mando qué petición a qué otra máquina?[1]. Así se definen **N** paquetes:

```
packets {
packet-1 : ???.???.???.??? -> ???.???.???.??? : ??? ?? via "???";
packet-2 : ???.???.???.??? -> ???.???.???.??? : ??? ?? via "???";
.
.
.
packet-N: ???.???.???.??? -> ???.???.???.??? : ??? ?? via "???";
}
```

Donde los campos corresponden a:

- La ip de origen

- La ip de destino
- El protocolo a utilizar. Los protocolos pueden ser: tcp, udp o any (los dos anteriores)
- El numero de puerto de destino del paquete. Un natural en el rango `[0, 65535]` ¿Qué servicio está solicitando? Suele estar estrechamente relacionado con el protocolo seleccionado.
- El último campo indica, entre `""`, la interfaz por la cuál se enviará el paquete. (?)

Sección rules

Acá es cuando entra en juego el firewall. Definimos en esta sección las cadenas de reglas a seguir por el firewall, discriminando si los paquetes: salen del firewall (**OUTPUT**), son enviadas al firewall (**INPUT**) o bien pasan por el firewall hacia otro destino (**FORWARD**)[2]. Así se define la sección de reglas:

```
rules {
    chain INPUT {
        // Reglas correspondientes a la cadena INPUT
    }
    chain FORWARD {
        // Reglas correspondientes a la cadena FORWARD
    }
    chain OUTPUT {
        // Reglas correspondientes a la cadena OUTPUT
    }
}
```

El órden de definición de las cadenas es arbitrario, no debe ser necesariamente el mismo que el exhibido. En cuanto a las reglas, cada regla se separa por un semicolon (`;`) y consiste en una sucesión de condiciones separadas por espacios (de la forma `-condicion ...`) terminadas por un `-do ACCION`, que especifica la acción a tomar sobre el paquete. Las acciones tomables sobre un paquete son:

- Aceptarlo (ACCEPT)
- Rechazarlo, avisando al remitente (REJECT)
- Rechazarlo, descartándolo "silenciosamente" (DROP)

Las condiciones que se pueden imponer para cada regla son:

- `-srcip ???.???.???.???, ...`
- `-dstip ???.???.???.???, ...`
- `-srcsubnet ???.???.???.???/??, ...`
- `-dstsubnet ???.???.???.???/??, ...`

- `-prot ???`. Donde el protocolo puede ser: `tcp`, `udp`, o `any` (los dos anteriores).
- `-srcp ??, ...`. Especifica el/los puertos del nodo origen que emitió el paquete. Los valores son números naturales en el rango `[0, 65535]`.
- `-dstp ??, ...`. Especifica el/los puertos a los cuales el emisor del paquete quiere acceder. Los valores son números naturales en el rango `[0, 65535]`.
- `-inif "???", ..."`. Especifica la/las interfaces por las cuales sale el paquete desde su origen. Los valores son cadenas de caracteres (las interfaces deben ir encerradas entre `""`).
- `-outif "???", ..."`. Especifica la/las interfaces a las cuales llega el paquete a su destino(?). Los valores son cadenas de caracteres (las interfaces deben ir encerradas entre `""`).

Adicionalmente, se agrega a la semántica la posibilidad de especificar una política por defecto (**default policy**). Esto es, ¿qué acción tomar si el paquete que pasa no coincide con ninguna de las reglas?

Al final de cada definición de una `chain`, opcionalmente se puede terminar con:

`-default ACCION;`

> NOTE
>
> Al igual que en **iptables**, el órden de definición de las reglas es importante. Las mismas se evaluarán en el mismo orden en el que fueron definidas, según la chain a la que pertenezcan (`INPUT`, `OUTPUT` o `FORWARD`).

Comentarios

Se pueden realizar **comentarios de línea** usando doble barra (`//`).

```
// Esto es un comentario, no se interpretará como código.
```

Ejemplo

Para aclarar todo lo antes mencionado, mostramos un ejemplo de archivo en `test.fws`:

```
// test.fws
network {

    device servidor-web {
        mac = "AA:BB:CC:DD:EE:FF";
        ip = 192.168.1.10;
        subnet = 192.168.1.0/24;
        interfaces = "ppp0";
```

```
    }

    device pc-lab {
        mac = "DD:EE:FF:11:22:33";
        ip = 192.168.9.1;
        subnet = 192.168.9.0/16;
        interfaces = "ppp0", "wlan0", "wlan1";
    }

     // se debe definir de manera obligatoria.
     device firewall {
        mac = "00:11:22:33:44:55";
        ip = 192.168.1.1;
        subnet = 192.168.1.0/24;
        interfaces = "eth0", "eth1";
    }
}

packets {
    p1 : 192.168.1.10 -> 192.168.1.1 : tcp 22 via "ppp0";
    p2 : 192.168.9.1 -> 192.168.1.10 : tcp 80 via "wlan1";
    p3 : 192.168.1.1 -> 8.8.8.8 : udp 53 via "eth1";
    p4 : 192.168.1.10 -> 192.168.9.1 : udp 67 via "ppp0";
}

rules {
    chain INPUT {
        -srcip 192.168.1.1, 10.0.1.2 -do ACCEPT;
        -dstip 10.0.0.1, 10.0.1.2 -dstp 80 -do DROP;
        -srcsubnet 192.168.1.0/24, 10.0.0.1/16 -do REJECT;
        -default DROP;
    }
    chain FORWARD {
        -prot udp -dstp 53 -outif "eth0" -do ACCEPT ;
        -prot tcp -dstp 80,443,53,22 -inif "eth1","wlan1" -do ACCEPT ;
    }
    chain OUTPUT {
        -dstip 8.8.8.8, 1.1.1.1 -do DROP;
        -default ACCEPT;
    }

}
```

> **TIP**
>
> Lo unico que debería ser tratado como una cadena de caracteres (es decir, encerrado entre `""`) son las direcciones MAC y las interfaces de red.

Instalacion

El proyecto corre sobre **stack**, una herramienta que permite gestionar proyectos en **Haskell**.

Para debian/ubuntu:

> sudo apt-get install stack

Para correr el proyecto, basta con navegar hasta el directorio `net-firewall-simulator` y correr:

> stack run

Lo cual lleva a una consola de comandos interactiva.

(completar)

Footnotes

1. Lo interesante será cuando dos nodos de subredes distintas se comuniquen. Convenimos que el firewall acepta automáticamente los paquetes de tráfico local (un firewall no simulado, directamente ni vería los paquetes porque no pasan por él) ↩

2. Para más información, consultar el man page de iptables (`man iptables`) o iptables-tutorial ↩