

Fuente: IA Stable Diffusion

LABORATORIO 7. SEGURIDAD DE RED

DEPARTAMENTO DE INFORMÁTICA. UNIVERSIDAD DE OVIEDO

Seguridad de Sistemas Informáticos | 2022 – 2023 (v3.1 "S-81 Isaac Peral")





CONTENIDO

Infraestructura de este laboratorio	3
Bloque 1: Protección de redes de un host	6
Firewall de filtrado de paquetes (PF) para un host: Administración de zonas con <i>firewalld</i>	7
Firewall de filtrado de paquetes (PF) para un host: Bloqueo de direcciones IP y redes	10
Reenvío de puertos en <i>firewalld</i>	10
Protección de conexiones salientes: filtrado de DNS con <i>PiHole</i>	12
Filtrado avanzado de conexiones salientes <i>PiHole</i> : eliminar más URL maliciosas	13
Bloque 2. Protección de Conexiones	15
<i>Fail2Ban</i>	16
Protección de SSH mediante <i>Fail2Ban</i>	16
Comprobación del estado de <i>Fail2Ban</i>	17
Prueba de <i>Fail2Ban</i>	18
Honeypots: <i>PortSentry</i>	18
Registrar y bloquear intentos de análisis	19
Revertir bloqueos	19
Bloque 3. Extremando la seguridad en las conexiones	20
VPN <i>WireGuard</i>	21
Configuración del servidor VPN	21
Configurar redes de servidores	23
Configuración del cliente VPN	23
Emparejamiento de clientes y servidores	24
Probar la conexión VPN	24
Insignias y Autoevaluación	26

AVISO

Este material forma parte de la asignatura “Seguridad de Sistemas Informáticos”, impartida en la *Escuela de Ingeniería Informática* de la *Universidad de Oviedo*. Es fruto del trabajo continuado de elaboración, soporte, mejora, actualización y revisión del siguiente equipo de profesores desde el año 2019

- Enrique Juan de Andrés Galiana
- Fernando Cano Espinosa
- Miguel Riesco Albizu
- José Manuel Redondo López
- Luís Vinuesa Martínez

Te pedimos por favor que **NO lo compartas públicamente en Internet**. No obstante, entendemos que puedas considerar este material interesante para otras personas. Por ese motivo, hemos creado una versión de este adaptada para que pueda cursarse de forma online, disponible gratuitamente para todo el mundo y que puedes encontrar en esta dirección: <https://ocw.uniovi.es/course/view.php?id=109>

A diferencia de esta versión, **la versión libre puedes promocionarla todo lo que quieras**, que para eso está 😊

GRACIAS POR TU COLABORACIÓN

INFRAESTRUCTURA DE ESTE LABORATORIO



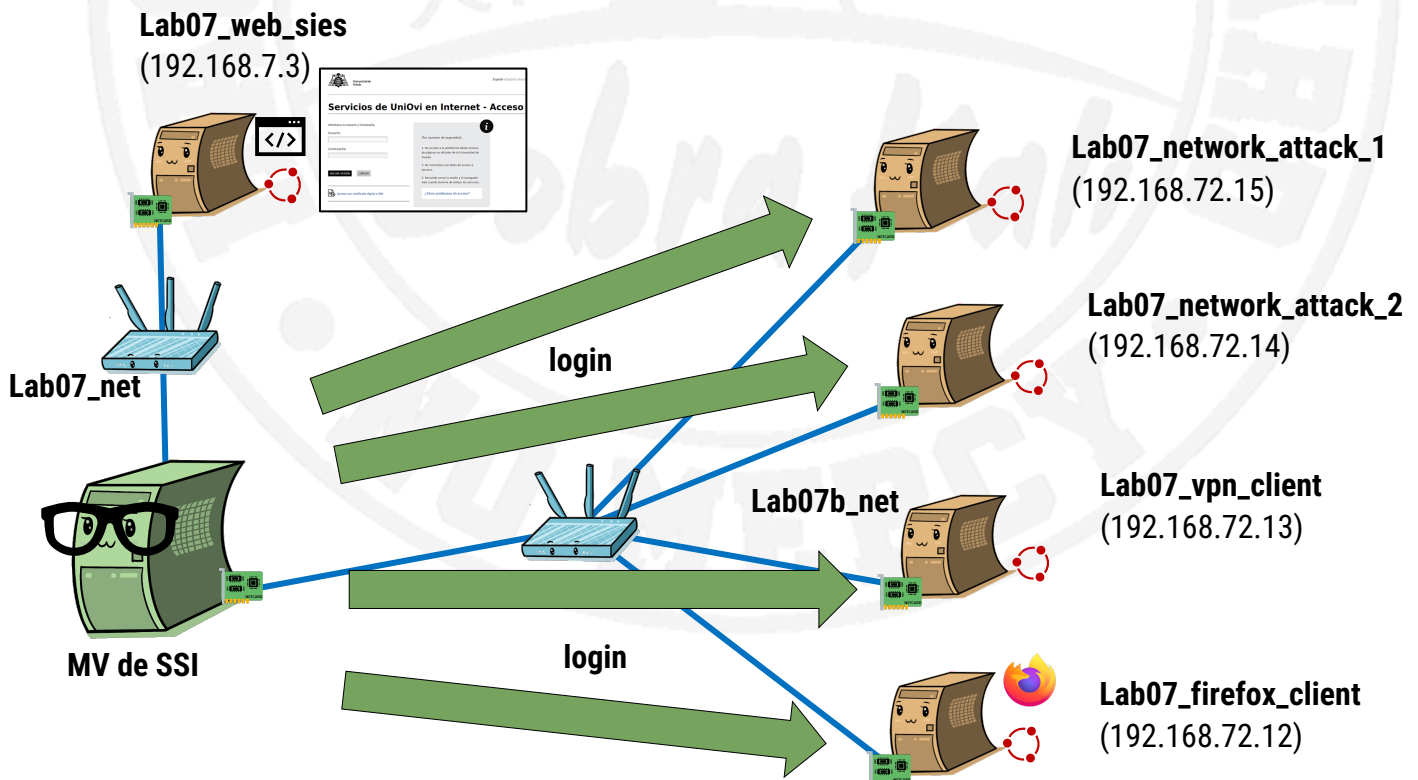
(NOTA: Para evitar problemas, te recomendamos encarecidamente que ejecutes esta infraestructura de laboratorio **en la máquina virtual base en lugar de en sus versiones con hardening** que puedas haber creado en laboratorios anteriores. En la vida real, deberías hacer lo contrario, pero las implementaciones reales también tienen un tiempo de validación para garantizar que todo funcione como se espera después del hardening.

AVISO: Algunos de vosotros habéis tenido problemas ejecutando Firefox desde un contenedor, algo que haremos en este laboratorio. Para evitarlos si te ocurren, sigue estos pasos:

- Abre el fichero `build_lab07.sh`
- Añádele la siguiente línea: `xhost + local:root`
- Construye el laboratorio como otras veces

La infraestructura de este laboratorio está compuesta por cinco contenedores comunicados a través de redes internas con la MV. La mayoría se usará de forma interactiva.

- Dos contenedores son contenedores de "ataque de red" y cuentan con las herramientas para probar el *firewall* y otro software de protección que vamos a ver en este laboratorio. La información que se escriba en el directorio `/etc/sysctl.d` se escribirá también en el directorio de la máquina virtual `volume_data/network_data_<N>/sysctl.d/`
- Un tercero que está diseñado para actuar como cliente VPN de *Wireguard*. La información escrita en el directorio `/etc/wireguard` se escribe en el directorio `volume_data/vpn_client/wireguard` de la máquina virtual.
- Un cuarto contenedor está habilitado para ejecutar aplicaciones GUI y tiene *Firefox* instalado.
- Finalmente, el último contenedor es un servidor web con la página *front-end* de la intranet de Uniovi comunicada con la MV con una red privada especial (`192.168.7.0/24`) y con IP fija.





(NOTA: Algunas de las operaciones de este laboratorio requieren conocer la dirección IP de los contenedores y las redes que están utilizando. Por favor, recuerda usar **ifconfig** dentro de los contenedores para conocer sus direcciones IP actuales)

Estos contenedores tienen instalado el software necesario para realizar las actividades del laboratorio. Sin embargo, varios de los productos que estamos a punto de utilizar son incompatibles con contenedores y, por lo tanto, **deben instalarse en la máquina virtual Ubuntu**. En este caso, los contenedores actuarán como "clientes atacantes" de la MV, por lo que podrás probar que funcionan sin necesidad de usar más elementos externos. Qué software está dirigido a los contenedores y cuál se va a instalar en las máquinas virtuales se indica claramente en cada actividad.



BLOQUE 1: PROTECCIÓN DE REDES DE UN HOST



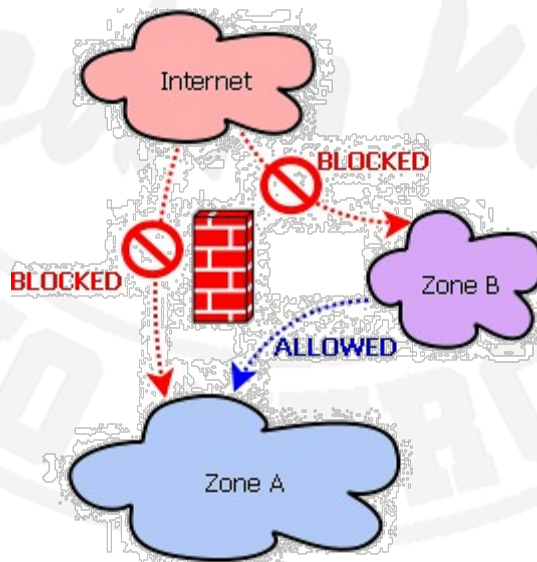
Firewall de filtrado de paquetes (PF) para un host: Administración de zonas con *firewalld*

Aplicación práctica: Necesitas separar tus interfaces de red en zonas con *firewalld* para poder asignarles diferentes configuraciones

Esta actividad te enseñará cómo usar *firewalld* para realizar operaciones de seguridad de red típicas que ilustren partes de la SNI (Secure Network Infrastructure) de la teoría. *firewalld* es un producto **más potente** que el *ufw* tradicional que viene con cada distribución de *Ubuntu*, sirve tanto para distribuciones *Debian* como *RedHat* y además lo usáis en *ASR*. No obstante, tener ambos habilitados a la vez causa problemas, por lo que primero **debemos asegurarnos de que *ufw* esté deshabilitado**: `sudo ufw disable`.

Una vez hecho esto, necesitamos instalar el software adecuado para usar y gestionar *firewalld*, consistente en el propio servicio de *firewall* y su GUI (*firewall-config*). Puedes instalar ambos de esta manera: `sudo apt install firewalld firewall-config`. Una vez hecho esto, **es muy importante reiniciar la máquina virtual**.

Una vez que la VM se haya reiniciado podremos trabajar con *firewalld*. Lo primero que debemos entender de este *firewall* es que distribuye las diferentes tarjetas de red que tengamos en la MV en **zonas**. Las zonas son conjuntos predefinidos de reglas que especifican qué tráfico se debe permitir en función del nivel de confianza en las redes a las que está conectado el equipo. Puedes asignar interfaces de red de la máquina que tiene *firewalld* a una zona. A partir de entonces, todas las máquinas conectadas a la misma red que cada una de esas tarjetas de red (asumiendo que mandan todo su tráfico a otras redes a través de dicha tarjeta, es decir, que la máquina con *firewalld* **actúa como puerta de enlace entre redes**), pertenecerán a dicha zona. Podrás así distribuir las máquinas por zonas gracias a *firewalld*, formando algo similar a lo que vimos en teoría con el SNI o este diagrama, que es una versión más sencilla. En ambos casos se cumple un principio: **el *firewall* hace de “policía” e “inspector de aduanas” de toda las conexiones**, poniéndose siempre “en medio”:



Fuente: <https://akm111.wordpress.com/2017/04/29/linux-firewalld-zones-and-services>

A continuación se muestran las zonas que incorporar *firewalld*:

- **docker:** Una zona especial que inicialmente contiene todas las redes creadas por nuestros contenedores *Docker*. La tenemos porque se crea cuando se instala *Docker* en el sistema.
- **Zonas donde las conexiones entrantes están prohibidas y solo se permiten conexiones salientes**
 - **drop:** Todas las conexiones entrantes se eliminan sin ninguna notificación



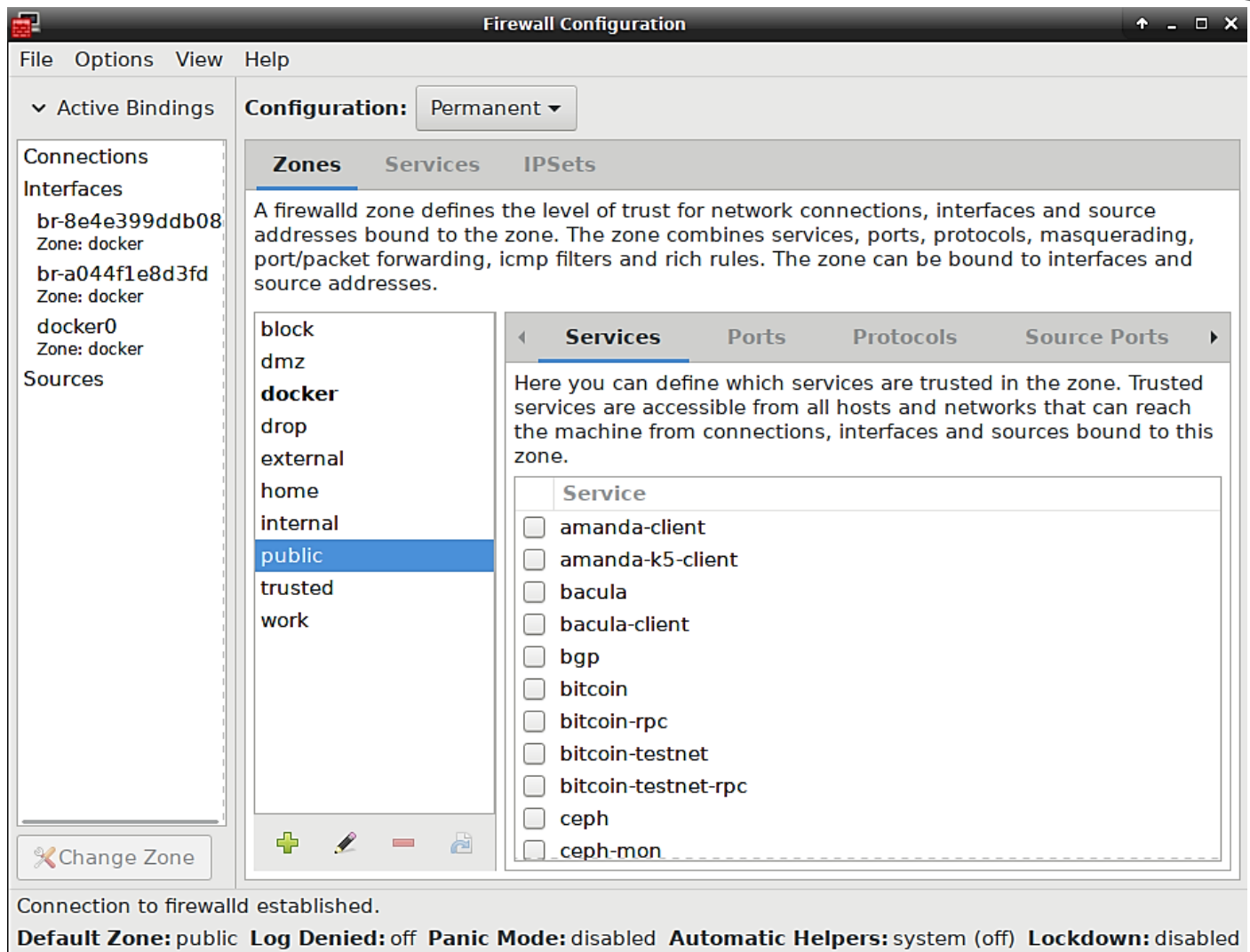
- **block:** Todas las conexiones entrantes son rechazadas (se notifica al cliente)
- **Zonas donde se permiten conexiones entrantes que previamente autoricemos nosotros. Como puede verse, están pensadas para diferentes situaciones y redes donde se puede encontrar la máquina que tiene `firewalld` instalado**
 - **public:** Para uso en **áreas públicas** no confiables. No confía en otros equipos de la red.
 - **external:** Para uso en **redes externas** con enmascaramiento NAT habilitado, cuando el sistema actúa como puerta de enlace o enrutador.
 - **internal:** Para uso en **redes internas**, cuando el sistema actúa como puerta de enlace o enrutador. Los otros sistemas de la red son generalmente de confianza.
 - **dmz:** Se utiliza para equipos ubicados en una **zona desmilitarizada o DMZ** (Tema 5 de teoría), que tendrán acceso limitado al resto de su red.
 - **work:** Utilizado para **máquinas del trabajo**. Otros equipos de la red son generalmente de confianza.
 - **home:** Utilizado para **máquinas domésticas**. Otros equipos de la red son generalmente de confianza.
- **Zonas donde no se restringe ninguna conexión**
 - **trusted:** Se aceptan todas las conexiones de red. Confía en todos los equipos de la red.

No tienes que entender todas estas zonas, pero necesitas saber a cambiar las interfaces entre ellas. La parte buena de usar zonas es que una vez que habilitas un servicio o puerto para una zona, **¡lo habilitas para todas las interfaces que pertenecen a esa zona al mismo tiempo!**

Vamos a usar el GUI *Firewall Config* para administrar la mayoría de las operaciones del *firewall*, pero también puedes hacer las mismas cosas a través de la línea de comandos:

- <https://computingforgeeks.com/install-and-use-firewalld-on-ubuntu/>
- <https://www.supportsages.com/everything-you-need-to-know-about-firewalld/>

Para hacer eso, primero ejecuta `sudo firewall-config` (recuerda usar `sudo`, por favor, o te dará un error) y cambia el *ComboBox* de **Configuration** a **Permanent**, como se muestra en esta imagen:



Desafortunadamente, la mayoría de vosotros verá que las tarjetas de red **eth0** y **eth1** no están asignadas a ninguna zona y no aparecen en la GUI. Podemos arreglar esto usando la línea de comandos, haciendo que aparezcan estas tarjetas de red, y pudiendo manipularlas usando la GUI a partir de ahora. Haz las siguientes operaciones para eso:

- Agregar **eth0** (la tarjeta que permite que la máquina virtual se conecte a Internet) a la zona "public": **sudo firewall-cmd --zone=public --change-interface=eth0**
- Agregar **eth1** (la tarjeta que permite que la máquina virtual se conecte a tu PC) a la zona "work": **sudo firewall-cmd --zone=work --change-interface=eth1**

Una vez hecho esto, asegúrate de que la zona **work** tenga habilitado el servicio **ssh** y prueba que **ssh** está funcionando desde tu PC.

Resultados esperados: Esta actividad finalizará cuando se puedan asignar zonas a las tarjetas de red en el **firewall** y activar y probar que se puede acceder a un servicio en la zona "work".

Firewall de filtrado de paquetes (PF) para un host: Bloqueo de direcciones IP y redes

Aplicación práctica: Necesitas bloquear IPs individuales o redes completas para que no puedan acceder a tu máquina

Uno de los usos más comunes de un *firewall* es permitir/bloquear el acceso desde ciertas IPs o redes a otras IPs y redes. Para practicar cómo hacer esto, sigue este procedimiento:

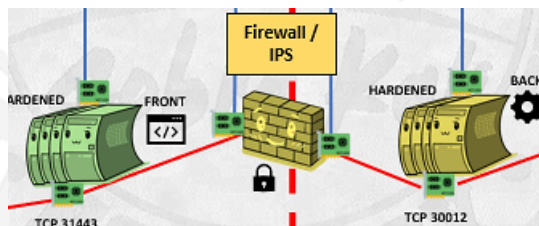
- Asegúrate de que puedes acceder desde tu PC a la máquina virtual a través de **ssh**
- Vete a la zona de "**work**" en la GUI del firewall y busca en la pestaña "**Rich Rules**".
- **Cree una nueva regla enriquecida que elimine (drop)** todas las conexiones al servicio **ssh** desde la IP **192.168.56.1** (normalmente la IP de tu PC en la red *host-only*) a la IP **192.168.56.10** (la IP de tu máquina virtual en la red *host-only*). Comprueba además todas las opciones que tiene la creación de reglas y piensa en sus posibilidades, especialmente las opciones de *limit*, *log* y *audit*.
- Prueba que ahora ya no puedes conectarte a través de **ssh**
- Modifica la regla anterior para bloquear toda la red **192.168.56.0/24**
- Prueba que aún no puedes conectarte a través de **ssh**
- Elimina la regla y prueba que ahora puede conectarte de nuevo.

Resultados esperados: Esta actividad finalizará cuando puedas bloquear una IP o red para acceder a un servicio en una zona de *firewall*.

Reenvío de puertos en firewall

Aplicación práctica: Necesitas "esconder" dónde está realmente un servicio (la IP y puerto de su máquina servidora real) redirigiendo su acceso a través de un firewall

El objetivo de este ejercicio es emular la siguiente parte del SNI:

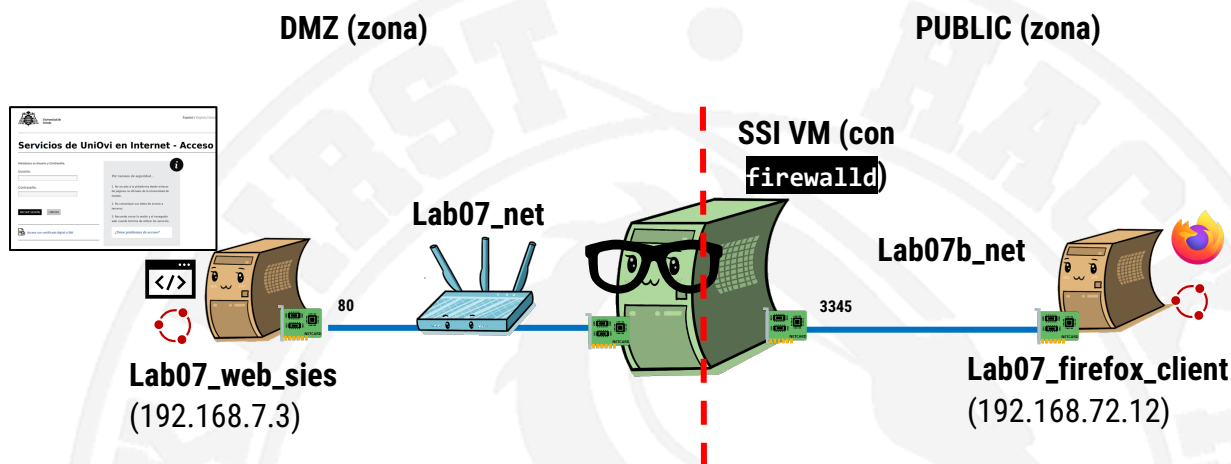


Tenemos dos máquinas situadas en diferentes zonas de red y, por lo tanto, no pueden comunicarse directamente. Entre ambas zonas hay un *firewall* que puede conectarse a ambas, y podemos indicar al *firewall* para que redirija una conexión realizada a uno de sus puertos en una de las zonas a otra máquina que está en una zona diferente: esto se llama **reenvío de puertos (port forwarding)**. La máquina en una de las zonas no sabrá quién es la máquina que atiende su solicitud en la otra zona, y la máquina que da el servicio nunca tendrá contacto directo con sus clientes: el *firewall* aísla a ambas. Para emular esto, vamos a utilizar los siguientes elementos de nuestra **infraestructura del Lab 7** para emular este comportamiento (ver diagrama):

- Entra en sesión en el contenedor **lab07_firefox_client** y anota su IP (**ifconfig**)

- Vete a la VM y localiza la tarjeta que está conectada a la red que usa el contenedor de *Firefox* buscando la que tiene una IP compatible con la que anotaste en el primer paso. Su nombre debería ser algo como **br-5c8ea19b05fb**
- Usando la GUI del *firewall*, coloque esta tarjeta de red en la zona "*public*".
- Ahora, localiza la tarjeta de red que sirve al contenedor **lab07_web_sies** (la que tiene una IP en la red **192.168.7.0/16**), que debería ser **192.168.7.3**.
- Coloca esta tarjeta de red en la zona "*dmz*".

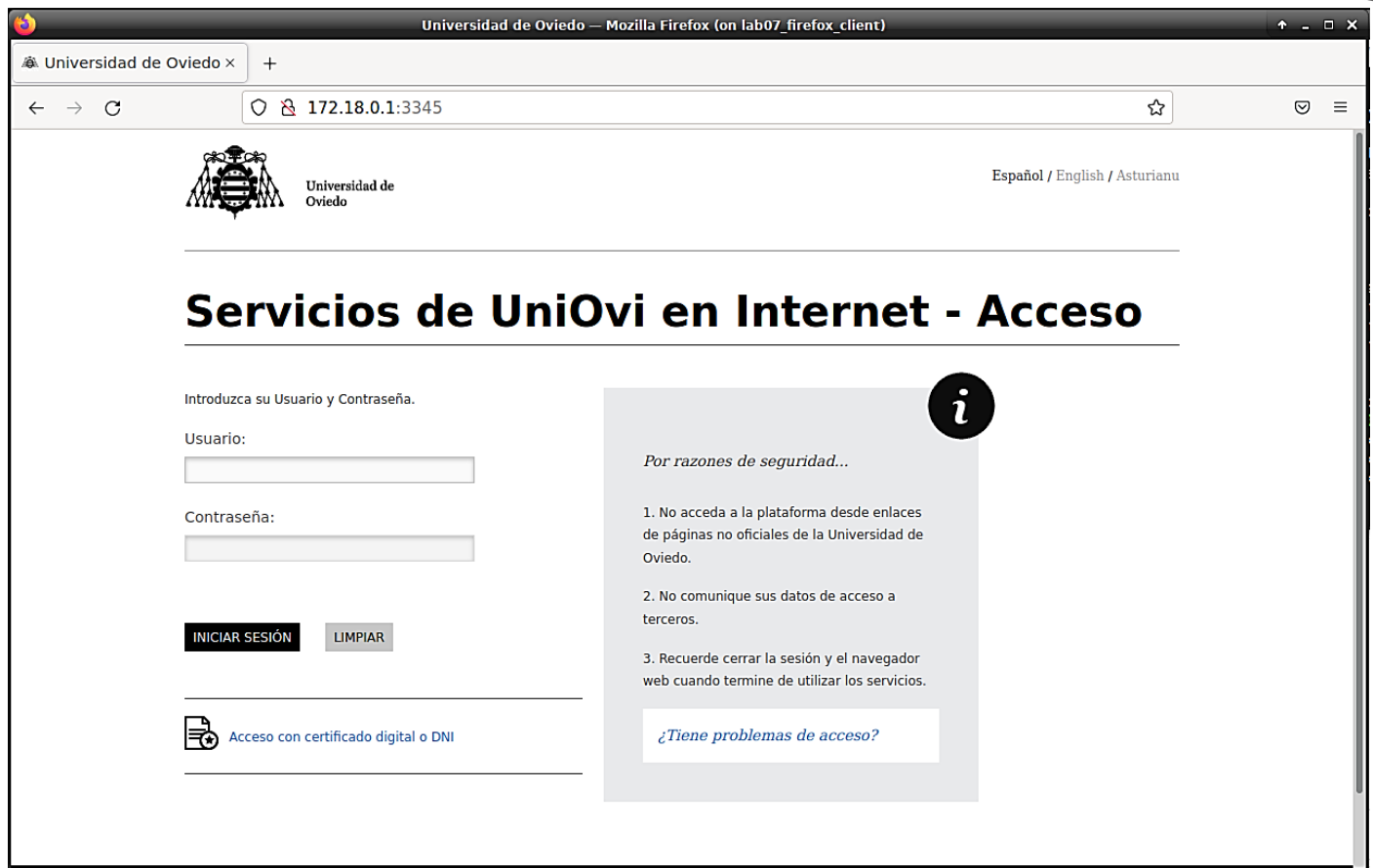
Este diagrama representa la estructura para este ejercicio:



Ahora has configurado dos contenedores en diferentes zonas y el *firewall* (tu máquina virtual) en el medio. Para poder comunicarlos mediante el reenvío de puertos, haz lo siguiente:

- Vete a la zona "*pública*" en la GUI y busca la pestaña "**Port Forwarding**" (dale a la flecha derecha para ver más pestañas)
- Crea una nueva regla de reenvío de puertos que redirija todas las conexiones al puerto **3345** al puerto **80** de la IP **192.168.7.3** (el contenedor **lab07_web_sies**). Si el firewall te pide que habilites el enmascaramiento, responde Yes.
- **Vuelve a cargar el firewall (Options – Reload firewall).**

Para probar esto, puedes ejecutar *Firefox* en la línea de comandos *bash* del contenedor **lab07_firefox_client**. La primera vez probablemente dará un error, **pero si lo reinicias, funcionará**. Ahora tienes un *Firefox* aislado de contenedor en la zona "pública". Navega a la IP de la máquina virtual en esta zona (si la IP del contenedor es **192.168.72.12**, la IP de la máquina virtual en esta red usa la misma red, pero termina en **1**) pero al puerto **3345** (Ejemplo: **192.168.72.1:3345**). Si todo está bien, deberías ver esta página web, que es justo la que tiene alojada la máquina "del otro lado" del firewall, pero...**¿ni sabes su IP ni el puerto real al que estás accediendo!**:



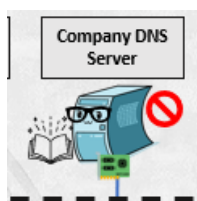
Con esto, has conectado una máquina en una red con una máquina a otra red a través de un *firewall*. Piensa en lo que esto significa con respecto a la seguridad (quién puede acceder a la segunda máquina, qué servicios se están exponiendo ...) y cómo podrías replicar esto para conectar un *front-end* de una aplicación con su *backend* si ambos se colocan en diferentes zonas de red.

Resultados esperados: Esta actividad finalizará cuando puedas comunicar un contenedor en la zona pública con un contenedor en la zona DMZ para que se pueda acceder a un *front-end* web colocado en la DMZ desde la zona pública mediante el reenvío de puertos. *¿Entiendes ahora qué es el aislamiento de redes?*

Protección de conexiones salientes: filtrado de DNS con PiHole

Aplicación práctica: Necesitas bloquear conexiones a dominios maliciosos conocidos

Te recomendamos encarecidamente que crees una instantánea de la máquina virtual antes de realizar este ejercicio. El objetivo de este ejercicio es mostrarte la importancia de controlar tu propio servidor DNS para filtrar el tráfico saliente potencialmente malicioso. Esto corresponde a esta parte de la teoría *Secure Network Infrastructure* (SNI), pero también lo puedes usar en casa 😊.



Haremos esto configurando un producto profesional, *PiHole* (<https://pi-hole.net/>), que te permite tener un servidor DNS que además filtra cualquier solicitud que reciba utilizando un enfoque **blocklist**: cualquier conexión realizada a una IP presente en la lista de bloqueo se anulará. *PiHole* permite la instalación en muchos sistemas, pero para este laboratorio lo hemos empaquetado en un contenedor. Vete a la carpeta **pihole** de los archivos de laboratorio y ejecuta el script **run_pihole.sh** (por favor, asegúrate de que *Apache2* no esté instalado en la máquina virtual y haz **sudo apt-get remove apache2** si lo está). Una vez termine de ejecutarse el comando, navega a la URL **127.0.0.1/admin** en el navegador de la VM para ver la pantalla de estadísticas de bienvenida de *PiHole*. La contraseña de administrador es **test123...**

Te recomendamos que navegues ahora a cualquier página web con publicidad (como www.elmundo.es) y dejes esa web abierta. Ahora que *PiHole* está funcionando, para empezar a filtrar conexiones solo tenemos que cambiar el DNS de nuestra tarjeta de red **eth0** (la que se conecta a Internet) a **127.0.0.1** (*PiHole* se ejecuta localmente). No te preocupes, no perderás la conexión a Internet, *PiHole* sabe qué hacer 😊 La siguiente imagen muestra los cambios que debes realizar para esto, **aunque también tendrás que cambiarlo en el fichero `/etc/resolv.conf`**:

```
GNU nano 2.9.3 /etc/netplan/01-netcfg.yaml Modified
1 network:
2   version: 2
3   ethernets:
4     eth0:
5       dhcp4: true
6       nameservers:
7         addresses: [127.0.0.1]
8
```

Vuelve a cargar ahora el sitio web anterior. ¿Ves alguna diferencia? ¿Qué pasará ahora con cualquier dispositivo que utilice este servidor DNS?

Resultados esperados: Esta actividad finalizará cuando puedas configurar el servidor de filtrado DNS *PiHole* para probar cómo cambia su navegación una vez que uses sus funciones.

Filtrado avanzado de conexiones salientes *PiHole*: eliminar más URL maliciosas

Aplicación práctica: Necesitas bloquear conexiones a aún más dominios maliciosos

Aunque *PiHole* filtra un gran número de sitios web maliciosos de serie, podemos darle más “potencia” utilizando las URL maliciosas recopiladas por **The Block List Project** (<https://blocklistproject.github.io/Lists/>). Sigue estas instrucciones para incorporar esta lista de IPs a *PiHole*:

- Copia el enlace de la lista más completa (**Everything**) en formato compatible con *PiHole* (<https://blocklistproject.github.io/Lists/everything.txt>).
- Agrega la URL a las listas de bloqueo de tu *PiHole* (**Login - Group Management-Adlists -Pega la URL de la lista en el campo "Dirección", agrega un comentario - Haz clic en "Add"**)
- Actualiza **Gravity** (**Tools-Update Gravity-Click "Update"**) para incorporar las nuevas URL a la lista de bloqueo. El proceso lleva un tiempo, **¡por favor espera y no navegues fuera de esta página!**



Navega de nuevo a cualquier web con la página web de estadísticas de *PiHole* abierta y observa cómo unas listas de bloqueo mucho más grandes están ahora en funcionamiento.

Resultados esperados: Esta actividad finalizará cuando puedas mejorar las listas de bloqueo de *PiHole* para bloquear las conexiones a más tipos de sitios web maliciosos.



BLOQUE 2. PROTECCIÓN DE CONEXIONES



Fail2Ban

Aplicación práctica: Necesitas bloquear las conexiones de las máquinas que tratan de acceder a tu servicio SSH sin éxito múltiples veces

Es una aplicación que **procesa los logs del sistema** para buscar indicios de ataques. Cuando se detectan, de acuerdo con las reglas que se le definan, **fail2ban** agrega una nueva regla al **firewall** que bloquea la IP desde la que se hizo el intento de ataque identificado. Esta regla bloquea la IP del atacante de forma temporal o permanente, dependiendo de la configuración. También se puede **notificar por correo electrónico** a un usuario que un ataque está en marcha (instalando **sendmail**, pero no lo usaremos en este laboratorio).

fail2ban se usa típicamente para **proteger ssh** pero tiene varias reglas predefinidas por defecto (llamadas **jails**) que también funcionan con otros servicios, como **Apache o Nginx** (servidores web). Técnicamente, cualquier servicio de red que genere un log puede ser protegido por **fail2ban** siempre que tenga configuradas las reglas adecuadas. Para hacer esto puedes crear tus propias reglas, pero eso excede el propósito de este laboratorio. Este programa no funciona dentro de contenedores, por lo que debes **instalarlo en tu máquina virtual**.

Protección de SSH mediante Fail2Ban

El uso de **fail2ban** con SSH requiere seguir estos pasos:

- Instálalo (el servicio se inicia automáticamente una vez instalado): **sudo apt install fail2ban**
- Asegúrate de que los servicios a proteger por **fail2ban** (SSH, HTTP, ...) no estén bloqueados por el **firewall** (¡no habría nada que proteger entonces! 😊).
- Modifica la configuración de **fail2ban** (archivo **/etc/fail2ban/fail2ban.conf**). Es **MUY** recomendable copiar este archivo a **fail2ban.local** y editar este nuevo archivo en lugar del **.conf** original. En este archivo puedes configurar cosas como el nivel de detalle del **log** (**loglevel**), dónde escribir la información de log (**logtarget**), etc. Usaremos la configuración predeterminada en este laboratorio.
- Modifica el archivo **/etc/fail2ban/jail.local** para habilitar o deshabilitar las **jails**. Para hacer eso, haz lo siguiente:
 - Si el archivo **jail.local** no existe, créalo copiando **jail.conf**
 - **Al principio, solo la jail sshd está activa por defecto.** El resto debe activarse manualmente agregando **enabled = true** después de cada nombre de **jail** que quieras activar (el que se encuentra entre **[]**).
 - Hay muchos otros comandos y parámetros (como IPs a ignorar), pero no los usaremos en este laboratorio, ya que queremos una implementación simple (pero efectiva) de **fail2ban**.
 - Cuando se activa una **jail**, los parámetros **bantime**, **findtime** y **maxretry** definen cómo se desautoriza una IP. Un **bantime** negativo significa un **bloqueo permanente**. Un **host es bloqueado por una jail** si activa sus condiciones con éxito más de **maxretry** veces a lo largo de **findtime** segundos.
 - Las **jails** disponibles se definen en una sección de este archivo. Como decíamos, **sshd** está habilitada por defecto, mientras que las demás deben activarse manualmente. Eso sí, esto solo se recomienda si realmente tenemos el servicio en marcha.



```
[sshd]
# To use more aggressive sshd modes set filter parameter "mode" in jail.local:
# normal (default), ddos, extra or aggressive (combines all).
# See "tests/files/logs/sshd" or "filter.d/sshd.conf" for usage example and det$
#mode    = normal
port     = ssh
logpath  = %(sshd_log)s
backend  = %(sshd_backend)s

[dropbear]
port     = ssh
logpath  = %(dropbear_log)s
backend  = %(dropbear_backend)s

[selinux-ssh]
port     = ssh
logpath  = %(auditd_log)s

#
# HTTP servers
#

[apache-auth]
port     = http,https
logpath  = %(apache_error_log)s

[apache-badbots]
# Ban hosts which agent identifies spammer robots crawling the web
# for email addresses. The mail outputs are buffered.
port     = http,https
```

- Cada comportamiento de *jail* se define en un archivo diferente en el directorio `/etc/fail2ban/filter.d`. Cada uno de estos archivos contiene **las expresiones regulares** que, al coincidir con una entrada del *log*, activan la regla. También definen modos de operación y otros parámetros. No es necesario examinarlos en este laboratorio, esto es solo para que lo sepas 😊
- Para este laboratorio **habilitaremos el modo agresivo para `sshd`** y garantizar el máximo modo de protección SSH.

Comprobación del estado de *Fail2Ban*

`fail2ban-client` puede usarse para gestionar `fail2ban` a través de las siguientes opciones:

- `start`: inicia `fail2ban`
- `reload`: Recarga la configuración de `fail2ban`
- `reload JAIL`: Reemplaza la configuración actual de la *jail* especificada por la que está en el archivo que se pasa
- `stop`: Detiene `fail2ban`
- `status`: muestra el estado actual de `fail2ban` y las *jails* activas (incluidos sus nombres)
- `status JAIL`: muestra el estado de la *jail* especificada. Es una opción muy importante ya que también muestra cuántas veces se ha activado y las IPs bloqueadas actualmente.

Cada vez que actives nuevas *jails* en el archivo anterior **debes volver a cargar `fail2ban`** y verificar cuántas *jails* están activas.

Prueba de Fail2Ban

Probar **fail2ban** con SSH es muy fácil: solo tienes que hacer varios intentos de inicio de sesión con contraseñas incorrectas dentro del tiempo que has configurado. Se recomienda hacerlo desde uno de los contenedores de la **infraestructura del Lab 7**. Una vez que has sido bloqueado, **usa los comandos anteriores para comprobar que la IP aparece realmente como prohibida**.

Para desbloquear la IP, puedes esperar el tiempo que has especificado (¡si no es infinito! 😊) o ejecutar:
fail2ban-client set <nombre de la jail> unbanip <ip>

Resultados esperados: Esta actividad finalizará cuando puedas proteger **ssh** utilizando el modo de protección agresivo de **fail2ban**. También puedes verificar que la protección funciona y revocar los bloqueos de IP creados por este programa.

Honeypots: PortSentry

Aplicación práctica: Necesitas bloquear conexiones de atacantes que están intentando escanearte

Los escaneos de puertos con *Nmap* se tratarán en el próximo laboratorio, y son una de las operaciones más comunes en cualquier actividad de *pentesting*. En este laboratorio veremos cómo detenerlos usando esta herramienta. **PortSentry puede detectar escaneos de puertos y bloquearlos**, por lo que la máquina que está escaneando no obtendrá información.

Existen varias formas de utilizar **portsentry**, pero vamos a usar una de las más interesantes: combinarlo con el *firewall* que desplegamos en una actividad anterior para **emular el comportamiento de un Honeypot**. Para ello, **dejaremos algunos puertos abiertos a propósito en el firewall**, aunque no haya ningún servicio que los escuche. Se detectarán los intentos de escaneo a través de estos puertos "falsamente abiertos" y la herramienta procederá a prohibir la máquina que hace el escaneo. Ten en cuenta que **portsentry** no puede escuchar en los puertos que tienen servicios reales que los están usando, ya que "ocupan" los puertos. Sigue estos pasos:

- Instala el servicio *PortSentry* en tu máquina virtual *Ubuntu* (no es compatible con contenedores): **sudo apt install portsentry**
- Comprueba que se está ejecutando: **sudo service portsentry status**
- Entiende los dos modos de **trabajo** de la herramienta (valores de los parámetros **TCP_MODE** y **UDP_MODE** en el archivo **/etc/default/portsentry**)
 - **Modo básico** (valores: **"tcp"**, **"udp"**): Escucha una lista estática de puertos predefinidos en el archivo de configuración. **No** utilizaremos este modo ya que la presencia de la herramienta se puede detectar fácilmente.
 - **Modo stealth avanzado** (valores: **"atcp"**, **"audp"**): Usa un *raw socket* para detectar escaneos y así la herramienta no puede ser detectada fácilmente: los puertos parecen tener el servicio "normal" en él, pero es **portsentry** quien está escuchando y detectando actividad sospechosa.
- Comprender las **posibles respuestas** a un intento de escaneo (valores de **BLOCK_UDP** y **BLOCK_TCP** en el archivo **/etc/portsentry/portsentry.conf**)
 - Solo hacer **log** del tráfico sospechoso en **/var/log/syslog** (valor: **"0"**) (valor por defecto)
 - **Bloquear a la máquina "infractora"** (valor: **"1"**)

- **Ejecutar un programa** como respuesta (valor: **"2"**). Esto nos da mucha flexibilidad y permite respuestas "salvajes" a los intentos de escaneo 😊, pero no lo usaremos aquí.

Registrar y bloquear intentos de análisis

Lo primero que hay que hacer para habilitar **portsentry** es habilitar los siguientes servicios en la zona de **firewall docker**: SSH, FTP y Telnet. Así tenemos un "cebo" para atraer los intentos de escaneo a nuestro servidor.

Una vez hecho esto, abre el fichero **/etc/default/portsentry** y pon **TCP_MODE** a **"atcp"** y **UDP_MODE** a **"audp"** para habilitar el modo **stealth avanzado**. Reinicia **portsentry** y haz un análisis **nmap** simple desde cualquiera de los contenedores de **infraestructura de Lab 7** a la máquina host (**nmap <IP>**). Una vez finalizado el análisis, comprueba el contenido de **/var/log/syslog** para ver si se han detectado los intentos de análisis.

Que se haga **log** con éxito de los intentos de escaneo es la condición para terminar de configurar **portsentry** según lo previsto: Abre **/etc/portsentry/portsentry.conf** y cambia los valores de los parámetros **BLOCK_UDP** y **BLOCK_TCP** a **"1"**. Reinicia **portsentry** y vuelve a realizar el mismo análisis.

Revertir bloqueos

Para revertir un bloqueo, es necesario hacer esto:

- Eliminar la IP de la máquina bloqueada de **/etc/hosts.deny**. Estar en este archivo impide cualquier comunicación desde esta IP a nuestra máquina.

```
GNU nano 2.9.3 /etc/hosts.deny
# /etc/hosts.deny: list of hosts that are _not_ allowed to access the system.
# See the manual pages hosts_access(5) and hosts_options(5).
#
# Example:  ALL: some.host.name, .some.domain
#           ALL EXCEPT in.fingerd: other.host.name, .other.domain
#
# If you're going to protect the portmapper use the name "rpcbind" for the
# daemon name. See rpcbind(8) and rpc.mountd(8) for further information.
#
# The PARANOID wildcard matches any host whose name does not match its
# address.
#
# You may wish to enable this to ensure any programs that don't
# validate looked up hostnames still leave understandable logs. In past
# versions of Debian this has been the default.
# ALL: PARANOID
ALL: 192.168.8.100 : DENY
```

- Eliminar la regla del **firewall** que rechaza (**REJECT**) todo el tráfico del cliente infractor: **sudo route wdel -host <IP del cliente infractor> reject**
- Podemos usar **netstat -rn** para comprobar que se ha eliminado la regla.

Resultados esperados: Esta actividad finalizará cuando seas capaz de proteger un host contra los intentos de escaneo **nmap** con **portsentry** y comprender el comportamiento del bloqueo que realiza, todo sin interferir con el **firewall** de la máquina. También debes verificar que la protección funciona y deshacer los bloqueos de IP creados por este programa.

BLOQUE 3. EXTREMANDO LA SEGURIDAD EN LAS CONEXIONES



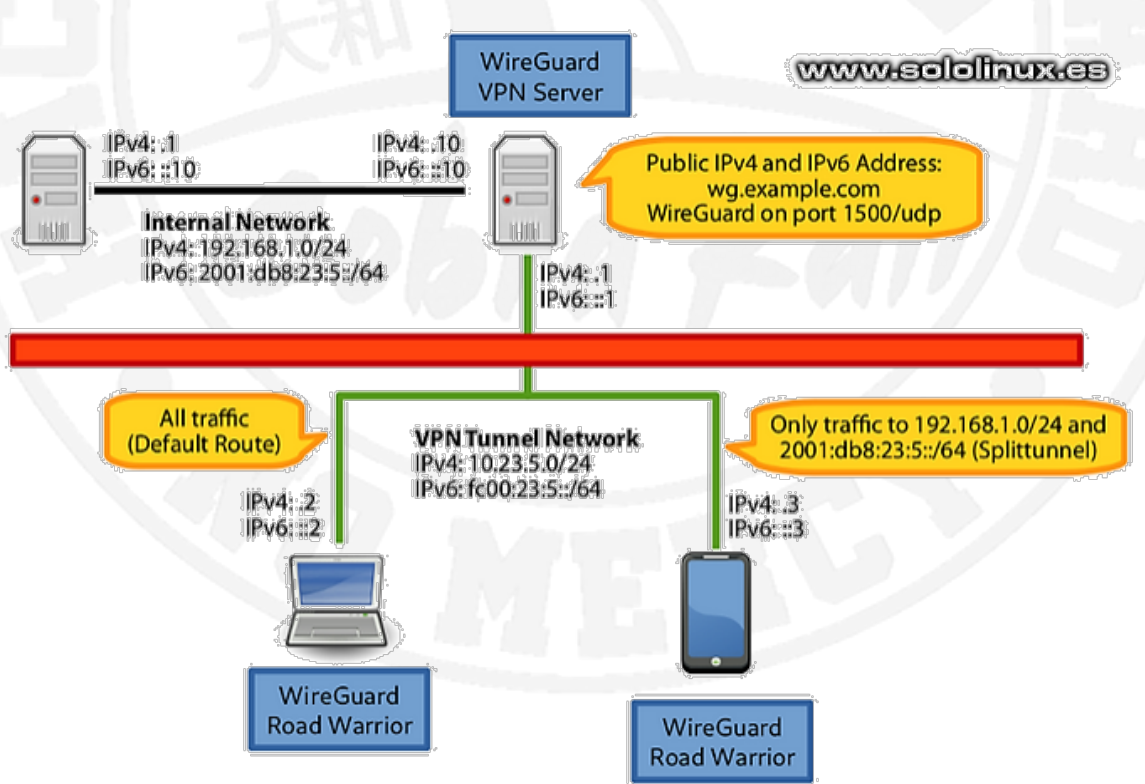
VPN WireGuard

Aplicación práctica: Necesitas poner en marcha tu propio servidor VPN

Una VPN es una red privada que garantiza la **comunicación privada de extremo a extremo, creando un “túnel” privado entre un cliente y un servidor incluso a través de redes públicas**. WireGuard es una de las mejores implementaciones de VPN disponibles que podemos usar para acceder a cualquier servidor con una IP pública desde cualquier parte del mundo de forma segura (y gratuita). Si nuestro servidor solo acepta conexiones vía VPN correctamente configuradas, **será invisible para el resto del mundo**. Esto coloca a un dispositivo cliente dentro de una red interna que definimos (de empresa, de casa...), accediendo a todos los recursos de esa red interna como si estuviera dentro de la misma.

Esta actividad se ha incluido porque es una de las soluciones adecuadas para el trabajo remoto seguro. Al tratarse de una actividad especial que puede complicarse si algo falla, su explicación se ha detallado más para que se pueda seguir fácilmente. Esto también te guiará en caso de que quieras crear una infraestructura como esta en una configuración real. También hemos incluido esto para ayudarte si quieres conectarte desde Internet a un servidor de tu casa de forma segura.

La siguiente imagen representa cómo funciona esta configuración: nuestros clientes (un móvil y un portátil) pueden conectarse a la red interna y obtener una IP dentro de ella gracias a una interfaz de red especial en un servidor VPN. Las VPN son posibles gracias a la **criptografía de clave pública**, por lo que esta es también una aplicación práctica del tema correspondiente de la asignatura. Esto significa que cada miembro de la VPN (el servidor y cada cliente) tendrá que generar un par de **claves públicas / privadas** para poder usarlas.



Configuración del servidor VPN

Nuestra máquina virtual **Ubuntu** actuará como un **servidor VPN** para el propósito de este laboratorio. Podemos instalar **WireGuard** como cualquier otro software típico de **Ubuntu**: `sudo apt install wireguard`. WireGuard incluye dos herramientas, **wg** y **wg-quick** para configurar y administrar las interfaces de **WireGuard**.

Para **generar las claves públicas y privadas** en el servidor necesitamos ejecutar: `wg genkey | sudo tee /etc/wireguard/privatekey | wg pubkey | sudo tee /etc/wireguard/publickey`. Los archivos se generarán en el directorio `/etc/wireguard`.

Una vez generados, necesitamos configurar el **dispositivo túnel** que enrutará el tráfico VPN. Para ello, creamos el siguiente archivo de configuración con un editor de texto para un dispositivo de túnel llamado `wg0` (aunque puedes poner el nombre que quieras) (`sudo nano /etc/wireguard/wg0.conf`):

```
[Interface]
Address = 10.0.0.1/24
SaveConfig = true
ListenPort = 51820
PrivateKey = <SERVER_PRIVATE_KEY (¡la que acabas de generar!)>
PostUp = iptables -A FORWARD -i %i -j ACCEPT; iptables -t nat -A POSTROUTING -o docker0 -j MASQUERADE
PostDown = iptables -D FORWARD -i %i -j ACCEPT; iptables -t nat -D POSTROUTING -o docker0 -j MASQUERADE
```

La configuración de la sección de interfaz tiene el siguiente significado:

- **Address:** una lista separada por comas de direcciones IP v4 o v6 para la interfaz `wg0`. Usa direcciones IP de un rango reservado para las redes privadas (`10.0.0.0/8`, `172.16.0.0/12` o `192.168.0.0/16`).
- **ListenPort:** el puerto en el que *WireGuard* aceptará conexiones entrantes. Recuerda esto para más adelante, ya **que debe estar abierto en el firewall del servidor**.
- **PrivateKey:** la clave privada generada por el comando `wg genkey` (`sudo cat /etc/wireguard/privatekey`)
- **SaveConfig:** si es `true`, el estado actual de la interfaz se guarda en el archivo de configuración cuando se apaga.
- **PostUp:** comando o *script* que se ejecuta antes de abrir la interfaz. En este ejemplo, estamos usando `iptables` para habilitar el enmascaramiento. Esto permitirá que el tráfico salga del servidor, dando a los clientes VPN acceso a Internet como en una configuración real típica (no la nuestra, ya que no estamos utilizando una interfaz "pública" real). Además, como usaremos un contenedor de la **infraestructura de Lab 7** como cliente, usaremos su nombre de la interfaz como la interfaz "pública" después de `-A POSTROUTING`. Aunque en la imagen aparece como `docker0`, en tu caso es más probable que el nombre sea diferente (algo así como `br-7b3b22d34e28`). Para saber esto, busca la interfaz en tu máquina virtual (`ifconfig`) que tenga una IP en la misma red que el contenedor `lab07_vpn_client` que estás utilizando. Asegúrate de cambiarlo por una interfaz pública real si estás usando esto en una configuración real.
- **PostDown:** comando o *script* que se ejecuta antes de desconectar la interfaz. Las reglas de `iptables` se eliminarán una vez que la interfaz esté inactiva.

Los archivos `wg0.conf` y `privatekey` no deben ser legibles para los usuarios normales. Utiliza `chmod` para cambiar sus permisos a `600`: `sudo chmod 600 /etc/wireguard/{privatekey,wg0.conf}`. Una vez hecho esto, habilita la interfaz `wg0` de forma que use los atributos especificados en el archivo de configuración: `sudo wg-quick up wg0`. El comando producirá una salida como esta:

```
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.0.0.1/24 dev wg0
[#] ip link set mtu 1420 up dev wg0
[#] iptables -A FORWARD -i wg0 -j ACCEPT; iptables -t nat -A POSTROUTING -o docker0 -j MASQUERADE
```




Ejecuta `sudo wg show wg0` para comprobar el estado y la configuración de la interfaz:

```
interface: wg0
  public key: r3imyh3MCYggaZACmkx+Cx1D6uAmICI8pe/PGq8+qCg=
  private key: (hidden)
  listening port: 51820
```

También puedes ejecutar `sudo ip a show wg0` para verificar el estado de la interfaz:

```
4: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state UNKNOWN group default qlen 1000
    link/none
    inet 10.0.0.1/24 scope global wg0
        valid_lft forever preferred_lft forever
```

Finalmente, para habilitar la interfaz de *WireGuard* en cada arranque puedes ejecutar el siguiente comando:
`sudo systemctl enable wg-quick@wg0`

Configurar redes de servidores

Esta VPN requiere NAT y esto entra en conflicto con una de las configuraciones que hicimos en un laboratorio anterior, por lo que debemos cambiarla para habilitar correctamente la funcionalidad VPN. Para que NAT funcione, necesitamos **habilitar el reenvío de IPs (IP Forwarding)**. Para ello abre el archivo `/etc/sysctl.conf` y modifica esta línea para que quede así: `net.ipv4.ip_forward=1`. Guarda el archivo y aplica el cambio con `sudo sysctl -p`. Además, **necesitamos abrir el tráfico UDP en el puerto 51820** (o el que configuremos en el archivo anterior) en el *firewall*.

Configuración del cliente VPN

Los clientes *Linux* también necesitan instalar el paquete `wireguard`. En nuestra [infraestructura de Lab 7](#) hay un contenedor **privilegiado** especial (consulta la sección de descripción de la infraestructura de laboratorio) que se creó precisamente para este propósito, e incluye este paquete y otros auxiliares para realizar las siguientes operaciones, por lo que no tienes que instalarlos.

El proceso para configurar un cliente *Linux* es prácticamente el mismo que en el servidor. Empieza por generar las claves públicas y privadas: `wg genkey | sudo tee /etc/wireguard/privatekey | wg pubkey | sudo tee /etc/wireguard/publickey`. A continuación, crea el archivo `wg0.conf` con el siguiente contenido (`sudo nano /etc/wireguard/wg0.conf`):

```
[Interface]
PrivateKey = <CLIENT_PRIVATE_KEY (la que acabas de generar)>
Address = 10.0.0.2/24

[Peer]
PublicKey = <SERVER_PUBLIC_KEY (la que se generó en la sección anterior)>
Endpoint = <SERVER_IP_ADDRESS (normalmente algo como 192.168.72.1sis, el host Docker)>:51820
AllowedIPs = 0.0.0.0/0
```

La configuración en la sección de interfaz tiene el mismo significado que cuando se configuró el servidor. Solo describiremos los siguientes. Si necesitas configurar clientes adicionales, simplemente repite los mismos pasos usando una dirección IP privada diferente:



- **Endpoint:** una IP o nombre de host del lugar al que quieres conectarte, seguido de dos puntos y, a continuación, un número de puerto en el que escucha el servidor remoto. En nuestro caso, esta será **la dirección de la máquina virtual de Ubuntu** en la interfaz de red del contenedor interno. Presta atención a la red de tu contenedor cliente (**192.168.72.1**) para poner la IP correcta aquí.
- **AllowedIPs:** una lista separada por comas de direcciones IP v4 o v6 desde las que se permite el tráfico entrante y a las que se dirige el tráfico saliente. Usaremos **0.0.0.0/0** porque estamos enrutando el tráfico y queremos que el servidor envíe paquetes con cualquier IP de origen.

Emparejamiento de clientes y servidores

El último paso es agregar la clave pública del cliente y su dirección IP al servidor: `sudo wg set wg0 peer <CLIENT_PUBLIC_KEY> allowed-ips 10.0.0.2`. Asegúrate de cambiar la **CLIENT_PUBLIC_KEY** con la clave pública que generaste en el equipo cliente y cambia la dirección IP del cliente si es diferente. Una vez hecho esto, vuelve a la máquina cliente para abrir la interfaz que hará el “túnel”.

Probar la conexión VPN

En el contenedor cliente *Linux* ejecuta el siguiente comando para abrir la interfaz: `sudo wg-quick up wg0`. Ahora deberías estar conectado al servidor *Ubuntu*, y el tráfico de tu máquina cliente debe enrutarse a través de él. Puedes comprobar la conexión con: `sudo wg`

```
root@18c398d86a51:/etc/wireguard# sudo wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.0.0.2/24 dev wg0
[#] ip link set mtu 1420 up dev wg0
[#] wg set wg0 fwmark 51820
[#] ip -4 route add 0.0.0.0/0 dev wg0 table 51820
[#] ip -4 rule add not fwmark 51820 table 51820
[#] ip -4 rule add table main suppress_prefixlength 0
[#] sysctl -q net.ipv4.conf.all.src_valid_mark=1
[#] nft -f /dev/fd/63
root@18c398d86a51:/etc/wireguard# wg
interface: wg0
  public key: M3VmkQTiBKu8I5HMkYhpEVvH9byTz94ZdHLgYbHj0jk=
  private key: (hidden)
  listening port: 48953
  fwmark: 0xca6c

peer: E5PZb/t/UCRGaR0o2GJMJR0lhvd4VNUUAN2TxfltzYE=
  endpoint: 172.17.0.1:51820
  allowed ips: 0.0.0.0/0
root@18c398d86a51:/etc/wireguard#
```

(NOTA: Si el último comando de la imagen es `iptables restore` en lugar de `nft -f`, el proceso puede darte un error. En este caso, puedes solucionarlo instalando el paquete `iptables` en el cliente)

Si examinamos las interfaces de red en el contenedor, `wg0` aparecerá con la IP local del cliente dentro de la red VPN.

```
wg0: flags=209<UP,POINTOPOINT,RUNNING,NOARP> mtu 1420
    inet 10.0.0.2 netmask 255.255.255.0 destination 10.0.0.2
    unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 1000
    (UNSPEC)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@18c398d86a51:/etc/wireguard#
```

Se puede realizar una prueba final haciendo ping al servidor desde el cliente utilizando la IP de la red VPN interna.

```
root@18c398d86a51:/etc/wireguard# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data:
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.774 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.172 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=2.01 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.294 ms
^C
```

Instala ahora el paquete **apache2** en la máquina virtual *Ubuntu* y verifica que puedes acceder a él desde la IP de la máquina servidora en la red VPN (**10.0.0.1**). Realiza un escaneo **nmap** desde el contenedor a esta IP para ver los resultados. Por último, intenta acceder a **10.0.0.1:3345** con **curl** para ver si puedes acceder a la página web del **lab07_web_sies** contenedor de un ejercicio anterior ahora. Si todas las pruebas tienen éxito, significa que la VPN está configurada correctamente.

Para detener el túnel, deshabilita la interfaz **wg0**: **sudo wg-quick down wg0**. Puedes encontrar información sobre cómo usar clientes de *Windows* aquí: <https://linuxize.com/post/how-to-set-up-wireguard-vpn-on-ubuntu-18-04/>





Resultados esperados: Esta actividad finalizará cuando puedas configurar con éxito una conexión VPN *WireGuard* entre un contenedor especial de la **infraestructura Lab 7** y la máquina virtual *Ubuntu*, pasando todas las pruebas de funcionamiento propuestas.

INSIGNIAS Y AUTOEVALUACIÓN



NOTA: Tienes una versión de esta tabla de insignias en formato editable disponible en el *Campus Virtual*. Puedes usar este archivo para crear un documento en el formato que desees y tomar notas extendidas de tus actividades para crear un log de lo que has hecho. Recuerda que el material que elabores se puede llevar a los exámenes de laboratorio.

Nivel de Insignia	Desbloqueado cuando	¿Desbloqueado?
	Sabes cómo permitir o denegar servicios/puertos de un <i>firewall</i>	
	Sabes cómo agregar interfaces de red a las zonas de un <i>firewall</i>	
	Puedes permitir o denegar conexiones desde direcciones IP individuales o redes	
	Puedes responder a esta pregunta: <i>¿Qué tipo de ataques podría mitigar potencialmente que un firewall ponga un límite de peticiones/tiempo a un servicio?</i>	
	Puedes responder a esta pregunta: <i>¿Cuál crees que es el propósito de registrar / auditar intentos de conexiones explícitamente prohibidas en el firewall?</i>	
	Puedes responder a estas preguntas: <i>¿Por qué crees que es útil bloquear a las máquinas que realizan escaneos? PISTA: ¿Escaneas máquinas como parte de tu rutina normal?</i>	
	Entiendes cómo realizar reenvío de puertos y como conectar máquinas en distintos segmentos de red separados por <i>firewalls</i>	
	Entiendes por qué el bloqueo de IPs vía DNS es útil. Puede responder a esta pregunta: <i>¿Cuál es la ventaja de usar PiHole para bloquear IPs en lugar de un complemento de un navegador que bloquee IPs maliciosas?</i>	
	Puedes responder a estas preguntas: <i>¿Crees que fail2ban reemplaza a un firewall? ¿o más bien lo complementa?</i>	
	Puedes responder a esta pregunta: <i>¿Qué tipo de ataques previene fail2ban?</i>	
	Puedes responder a esta pregunta: <i>¿Crees que un bloqueo temporal es suficiente para prevenir un intento de ataque DoS?</i>	
	Puedes responder a estas preguntas: <i>¿Por qué crees que dejar los puertos abiertos a propósito puede ser una medida útil de detección de escaneos? Por lógica, ¿qué puertos dejarías abiertos de esta manera?</i>	
	Puedes responder a estas preguntas: <i>¿Qué pasa cuando una máquina está bloqueada por portsentry? ¿La máquina bloqueada puede ponerse en contacto con la que bloquea usando cualquier servicio o no?</i>	
	Puedes responder a estas preguntas: <i>¿Afectan los bloqueos de portsentry a la velocidad de escaneo? Si la respuesta es sí, ¿crees que esto también se puede utilizar como medida de seguridad?</i>	
	Puedes responder a esta pregunta: <i>¿Cuántas cosas crees que puedes hacer para proteger las conexiones ssh desde el punto de vista de la red? (piensa en combinar técnicas que vimos aquí y laboratorios anteriores)</i>	
	Puedes protegerte de los intentos de ataque a ssh (y ataques similares a otros servicios que hacen) con <i>jails</i> predefinidas de fail2ban .	

	Puedes habilitar una "trampa de escaneo de puertos" en una máquina sin entrar en conflicto con el comportamiento normal del <i>firewall</i> . Puedes responder a esta pregunta: <i>si puedes recopilar las IP bloqueadas, ¿qué crearías con ellas?</i>	
	Puedes implementar una conexión VPN de <i>Wireguard</i> entre un contenedor y la máquina virtual host.	
	Si tenemos servidores web en una red interna diferente de nuestra empresa, y desplegamos con éxito una conexión VPN como la que vimos en este laboratorio, puedes responder a esta pregunta: <i>¿qué característica de firewall crees que podrá proporcionar acceso a estos servidores web fácilmente?</i>	
	Protección contra las redes oscuras: Tu conocimiento sobre la seguridad de red te permite implementar un servidor fortificado capaz de resistir los intentos de ataque de red más comunes contra los servicios típicos y como usuario final	

