Práctica 4

Los objetivos de esta práctica son los siguientes:

- Entender e implementar un sistema distribuído no-trivial con diferentes usuarios y niveles de acceso.
- Entender e implementar mecanismos de protección de intrusos.
- Entender e implementar mecanismos de detección de intrusos.
- Entender e implementar mecanismos de logging y auditoría.

Enunciado

En esta práctica vamos a implementar un sistema distribuido basándonos en la práctica 3 que haga uso de los mecanismos de seguridad que hemos visto en teoría. Este sistema distribuido tiene 2 componentes que se describen a continuación.

Servicio de base de datos

Ahora, nuestro servicio de base de datos tiene que reestructurarse de forma que pueda escalar para ser un servicio en la nube y exista personal que se encargue de su mantenimiento. Para ello, la aplicación se divide en 3 partes:

- `auth`: un servicio de autenticación que implementa todo lo relacionado con los usuarios y los tokens.
- `file`: un servicio de almacenamiento que implementa todo lo relacionado con la gestión de los archivos almacenados.
- `broker`: un servicio que recibe las peticiones de los usuarios y redirige las llamadas a `auth` o a `file` dependiendo de cuál sea la petición. De esta forma:
  - `/version` la atiende el propio `broker`.
  - `/login` y `/signup` las redirigirá a `auth`.
  - El resto las redirige a `file`.

Requisitos Como requisitos de implementación del servicio de la base de datos se encuentran los siguientes:

- Cada nuevo componente tiene que estar en su propio nodo.
- Cada nuevo componente debe comunicarse con el resto utilizando una API REST.
- La comunicación de todos los componentes debe hacerse por HTTPS.
- Toda petición de usuario debe pasar por el `broker` y éste debe proporcionar exactamente la misma API que la definida en la práctica 3.
- Es posible implementar nuevos endpoints en `auth` o `file`, pero no deben poder ser accesibles por los clientes externos.

- Las pruebas y clientes creados en la práctica 3 deben funcionar exactamente igual, sin ningún tipo de modificación.
- Se debe proporcionar un mecanismo automático (script, Makefile, etc.) que:
  - Cree todos los recursos necesarios.
  - Arranque todos los nodos del sistema.
  - Pare el sistema y destruya todo lo construído.
  - Corra pruebas automáticas.

Acceso SSH al personal

Además, se debe gestionar el acceso SSH al personal en todas las máquinas. Hay 2 usuarios:

- `dev` es un usuario desarrollador que sólo tiene acceso a una máquina de trabajo que llamaremos `work`. Este usuario *no* tiene acceso a `sudo`. El *único* nodo al que tiene acceso es a `work`.
- `op` es un usuario operador que tiene acceso a *todas* las máquinas del sistema y que además puede ejecutar `sudo` *sin necesidad de introducir ninguna clave*.

Requisitos Como requisitos de implementación del acceso SSH al personal se encuentran los siguientes:

- El acceso de `root` por SSH debe ser deshabilitado.
- Sólo se permite acceso utilizando cifrado asimétrico (clave pública/privada de SSH).
- El personal `dev` tiene a su disposición la máquina `work` a la que *únicamente* pueden llegar a través de un nodo de salto (*jump host*) llamado `jump`.
- El personal `op` puede acceder a cualquier máquina. Sin embargo, sólo pueden hacerlo desde la máquina `work`, que previamente debe ser accedida usando `jump` como intermediario. Por ejemplo, si `op` quiere acceder a `auth` primero debería acceder a `work` y después saltar a `auth`.
- En la máquina `jump` existen los usuarios `op` y `jump`. Sin embargo, sólo `jump` puede usarse desde el exterior. Por ello, tanto los usuarios `dev` y `op` tienen que usar el usuario `jump` para poder iniciar el primer salto al sistema.

Política de seguridad de la red

- *Cada nodo* implementará un cortafuegos utilizando `iptables`.
- Las políticas por defecto de las *chain* para la tabla *filter* deben ser:
  - `INPUT -> DROP`.
  - `FORWARD -> DROP`.
  - `OUTPUT -> ACCEPT`.

- ICMP está permitido en todas las direcciones.
- Todos los nodos deben ser capaces de actualizarse con los repositorios de Debian, por lo que DNS y HTTP debe estar permitido en todos ellos. HTTPS también está permitido en todos los nodos.
- Se definen 3 redes diferentes:
  - `dmz`: la red donde se pondrán los servicios que tienen que que ser contactados y accesibles desde el exterior (`broker` y `jump`)
  - `srv`: la red donde se pondrán los componentes del servicio `auth` y `file`.
  - `dev`: la red donde se pondrán los servicios para el personal. En este caso, el nodo `work`.
- En el esquema de red se definen los rangos y las IPs para cada uno de los nodos y las redes que se deben implementar en la práctica.
- Todo el tráfico de entrada y de salida va por el host.
- Desde el host *sólo* se puede acceder a los nodos a través del `router`. Aunque `docker` permite acceder a cada uno de ellos directamente, se debe *deshabilitar* esta opción.
- Todos los nodos deben tener activo el sistema de logs estandard `rsyslog`.

¿Qué se pide?

1. Implementar el sistema distribuido descrito anteriormente utilizando contenedores Docker.
   - Se proporcionará un `Makefile` que tendrá 3 objetivos:
     - `build`: para crear todos las imágenes necesarias y las redes necesarias.
     - `containers`: para lanzar todos los contenedores en el orden adecuado.
     - `remove`: para y borra todos los contenedores en marcha, y elimina todas las redes creadas.
     - `run-tests`: ejecuta una batería de pruebas que ejercite toda la API. Las pruebas pueden implementarse en Bash o en Python.
2. `README.pdf` explicando brevemente el proyecto, como instalarlo y cómo ejecutarlo. Debe contener las instrucciones necesarias para que cualquier profesional, sin necesidad de conocer cómo está implementado, pueda ejecutarlo localmente sin problemas.

Las prácticas entregadas se probarán con `curl` para su evaluación mediante pruebas automáticas, por lo que es muy importante implementar la API tal y como se especifica. Además, se examinarán las reglas definidas en `iptables`,

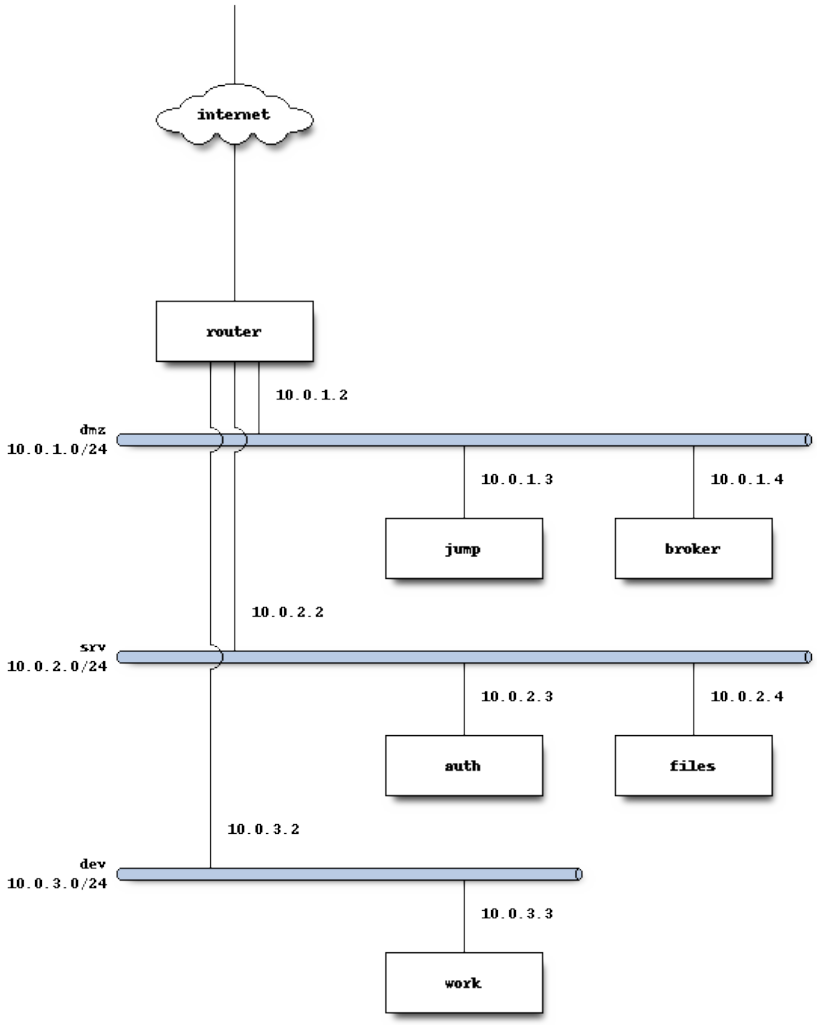

Figure 1: Esquema de red

permisos de los archivos, y demás aspectos que influyen en la seguridad del sistema distribuido.

¿Qué se valora?

1. La creación de un repositorio en GitHub privado y el desarrollo de la práctica en el mismo.
2. El grado de automatización en el despliegue y configuración de los contenedores.
3. La claridad y estructuración del código de forma que sea fácil de seguir y de leer.
4. Extras que aumentan la nota:
    - Política por defecto de `OUTPUT` a `DROP` en vez de `ACCEPT`.
    - Instalación y configuración de `fail2ban` en el caso de múltiples intentos fallidos de login al servicio durante un periodo de tiempo.
    - Instalación y configuración de un sistema de detección de instrusos (`snort`, `tripwire`, etc.).
    - Centralización de los logs generados por `rsyslog`. Esto se puede conseguir configurando todos los `rsyslog` de forma que envíen los logs a un nodo de la red `dev` (por ejemplo, un nodo llamado `logs`).
    - Cualquier otra propuesta acordada con el profesor.