

Servicios seguros en producción: SSH y Nginx

Seguridad Avanzada en Redes - Grado en Ingeniería Telemática

Olga Domingo Muñoz

19 de diciembre de 2024

Índice

1. Introducción	3
2. Entorno de trabajo	4
2.1. Uso de los servicios de Azure	4
3. Configuración de servicios	5
3.1. Configuración segura de SSH	5
3.1.1. Cierre de puertos	5
3.1.2. Autenticación por Clave Pública	6
3.1.3. Autenticación por Contraseña (2FA)	7
3.1.4. Autenticación con TOTP (3FA)	8
3.1.5. Auditoría SSH	9
3.1.6. Protección contra ataques de fuerza bruta con Fail2ban	11
3.1.7. Limitación del número de sesiones SSH con iptables	12
3.2. Configuración de Nginx	13
3.2.1. Instalación y configuración básica	13
3.2.2. Implementación de CertBot	14
3.2.3. Archivo de configuración principal	16
3.2.4. Archivo de configuración del servidor virtual labsec-vm	19
3.2.5. Locations	21
3.2.6. Protección con Fail2ban	23
3.2.7. Firewall ModSecurity con reglas OWASP	28
3.2.8. Administración de tareas con Cron	31
4. Puesta en producción	32
4.1. Manual para el acceso remoto al servicio	32
5. Conclusiones y futuras mejoras	34
5.1. Resumen de logros	34
5.2. Propuestas de mejora	35
6. Anexo	36
6.1. Referencias y enlaces consultados	36

1. Introducción

En esta práctica se pretende realizar la puesta en producción segura de los servicios SSH y Nginx siguiendo las técnicas de configuración estudiadas durante el curso. El objetivo es garantizar que ambos servicios estén accesibles desde Internet de manera controlada y protegida, cumpliendo con las mejores prácticas de seguridad.

Para ello, se ha configurado SSH aplicando medidas de seguridad avanzadas, y se ha desplegado Nginx, asegurando su correcto funcionamiento y protección. Además, se ha hecho uso de una máquina virtual (VM) en Azure como entorno de despliegue del servidor, permitiendo simular una infraestructura de producción accesible desde Internet.

Se contempla aplicar técnicas adicionales para reforzar la seguridad, como la configuración de un firewall, la implementación de herramientas como Fail2Ban, y el uso de métodos avanzados de endurecimiento del sistema. Estas medidas garantizan una protección de los servicios expuestos a Internet.

2. Entorno de trabajo

2.1. Uso de los servicios de Azure

Para realizar la práctica, se ha creado una máquina virtual (VM) en la plataforma **Azure**, donde se van a desplegar los servicios SSH y Nginx. La VM utiliza **Ubuntu Server 24.04 LTS** como sistema operativo y cuenta con una IP pública estática para permitir el acceso desde Internet, la **80.199.93.120**.

El nombre de dominio asociado es **labsec-vm.francecentral.cloudapp.azure.com**.

Una vez creada la máquina, se realizó la conexión inicial y se actualizó el sistema operativo con las últimas versiones de paquetes disponibles. Con este entorno configurado, podemos proceder a la instalación y configuración segura de los servicios SSH y Nginx.

3. Configuración de servicios

3.1. Configuración segura de SSH

3.1.1. Cierre de puertos

Durante la creación de la VM, se establecieron reglas para permitir únicamente la entrada de los puertos necesarios: 22 para SSH y 80/443 para HTTP/HTTPS. Todos los demás puertos quedaron bloqueados como medida de seguridad.

Reglas de puerto de entrada (6)				
300	⚠ SSH	22	TCP	Cualquiera
320	HTTPS	443	TCP	Cualquiera
340	HTTP	80	TCP	Cualquiera

Figura 1: Reglas de puertos de entrada

Al realizar un escaneo con **nmap**, se puede verificar que solo estos puertos están abiertos:

```
azureuser@vm:~$ nmap 20.199.93.120
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-12-18 16:23 UTC
Nmap scan report for 20.199.93.120
Host is up (0.0019s latency).
Not shown: 997 filtered tcp ports (no-response)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
443/tcp   open  https
```

Figura 2: Escaneo con *nmap*

Lo mismo usando la herramienta **netstat**, donde podemos ver los 3 puertos permitidos junto al puerto 53, destinado a la resolución de nombres de dominio. Esto se debe a que está abierto solo para consultas DNS internas (desde la misma red), como podemos ver en el segundo comando ejecutado.

```
azureuser@vm:~$ netstat -tnlp
(No info could be read for "-p": geteuid()=1000 but you should be root.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      PID/Program name
tcp        0      0 127.0.0.54:53           0.0.0.0:*              LISTEN     -
tcp        0      0 127.0.0.53:53           0.0.0.0:*              LISTEN     -
tcp        0      0 0.0.0.0:80             0.0.0.0:*              LISTEN     -
tcp        0      0 0.0.0.0:443            0.0.0.0:*              LISTEN     -
tcp6       0      0 :::22                :::*                  LISTEN     -
tcp6       0      0 :::80                :::*                  LISTEN     -
tcp6       0      0 :::443               :::*                  LISTEN     -
azureuser@vm:~$ sudo lsof -i :53
COMMAND   PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
systemd-r 474 systemd-resolve 14u  IPv4    4413      0t0  UDP _localdnsstub:domain
systemd-r 474 systemd-resolve 15u  IPv4    4414      0t0  TCP _localdnsstub:domain (LISTEN)
systemd-r 474 systemd-resolve 16u  IPv4    4415      0t0  UDP _localdnsproxy:domain
systemd-r 474 systemd-resolve 17u  IPv4    4416      0t0  TCP _localdnsproxy:domain (LISTEN)
```

Figura 3: Escaneo con *netstat*

3.1.2. Autenticación por Clave Pública

Para garantizar un acceso seguro al servidor desplegado en Azure, se implementó autenticación por clave pública usando el algoritmo RSA. Durante la creación de la máquina virtual, Azure generó esa clave pública asociada al usuario, en el archivo `/.ssh/authorized_keys` del servidor.

Este método ofrece una gran seguridad frente a ataques de fuerza bruta, garantizando un acceso eficiente y protegido al servidor.

```
azureuser@vm:~/ssh$ ls
authorized_keys id_rsa id_rsa.pub known_hosts
azureuser@vm:~/ssh$ sudo more authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAgQCe6n8ULF9uQUKzoqr86c0w8UDp+URiTzXwTIlpaAmjLUMIeT5Zs/yT4vK0ePpn9//+1scYw19kYWj8xz6bDdIQwKNZ4Z
evmmzvvU+9XyS16AN+UyCK3kAK33zjp00yMM5v86ID/SR7pCENBcX8ozwfql9p5qRlfZCRcailuc4JUQFd4Uz4/1DLGkMTEwLUQM72Vz7NuFr6iZPUDlzVA1GocUBcBQi
BLS810KT98m+LFJLmJze4/puStG7mbxzj1xmGOS0uiXFUZPnOfA/tGFV8awgJJUz1HzbDcq4ogyd4y37Vsdcx5h0CshUhd8s/WdaZXneSH5nJX++ATSu+nJleIEAnRiBLZxGY
3yQhBvZf0qwUH5sSXJDdo5tyxX0Ey4FJxajqVpxp5mUC+QhQkwB4IM50B+N3KRMbSCRSeSqbpp4tcZ9CrQ0v6SWIukQl0Rg0ufNUOimW9XQCC77xbHfx8y5wGwcZjxuoH2PNIL
cIYjD7/+kETV2tAXQIE= generated-by-azure
```

Figura 4: Claves usada

En el apartado *"Puesta en producción"* se podrá ver cómo aplicar este factor de autenticación a la hora de acceder de forma remota al servicio.

3.1.3. Autenticación por Contraseña (2FA)

Para mejorar la seguridad, aplicaremos también la autenticación por contraseña.

Esta se encuentra cifrada con un algoritmo de seguridad. Esto evita que sea visible en caso de acceso no autorizado al archivo, protegiendo así el sistema y los datos de los usuarios.

En el apartado *"Puesta en producción"* se podrá ver cómo aplicar este factor de autenticación a la hora de acceder de forma remota al servicio.

3.1.4. Autenticación con TOTP (3FA)

Para aumentar la seguridad de nuestro servidor, hemos añadido autenticación con TOTP (Time-based One-Time Password) a través de **Google Authenticator**. Esto significa que, además de ingresar la clave pública (el primer factor de autenticación) y la contraseña (segundo factor de autenticación), los usuarios deberán proporcionar un código temporal generado por la aplicación Google Authenticator instalada en sus dispositivos móviles.

Este código cambia cada 30 segundos, lo que añade una capa adicional de protección.

Para instalar el tercer factor de autenticación, primero se instaló el paquete *libpam-google-authenticator* en el servidor. Luego, se configuró su clave secreta utilizando el comando *google-authenticator*, lo que generó un código QR y claves de respaldo. Este código QR se escaneó con la aplicación móvil de Google Authenticator para sincronizar el dispositivo con el servidor.

Finalmente, se modificó el archivo PAM correspondiente (*/etc/pam.d/sshd*) para habilitar la autenticación basada en TOTP, y se ajustaron las configuraciones en */etc/ssh/sshd_config* para requerir esta autenticación como un paso adicional al iniciar sesión.

```
#AUTENTICACION GOOGLE AUTH
ChallengeResponseAuthentication yes
AuthenticationMethods publickey,keyboard-interactive
```

Figura 5: *sshd_config*

También se cambiaron los parámetros *UsePAM* y *KbdInteractiveAuthenticacion* a *yes*.

```
auth required pam_google_authenticator.so nullok
```

Figura 6: Archivo PAM *sshd*

En el apartado *"Puesta en producción"* se podrá ver cómo aplicar este factor de autenticación a la hora de acceder de forma remota al servicio.

3.1.5. Auditoría SSH

Como parte del proceso de mejora de la seguridad del servidor, realizamos una auditoría SSH utilizando la herramienta **ssh-audit**. Esta herramienta nos permitió analizar la configuración actual del servidor SSH, identificar vulnerabilidades y determinar las configuraciones inseguras que debían ser corregidas.

- Estado inicial de *sshd_config*:

Al ejecutar la auditoría inicial con ssh-audit, se detectaron debilidades en la configuración del servidor SSH como algoritmos obsoletos o inseguros y recomendaciones faltantes, ya que se utilizaban configuraciones predeterminadas que no cumplían con las mejores prácticas actuales.

```
azureuser@ve:~$ sudo ssh-audit 20.199.93.120
# general
(general) banner: SSH-2.0-OpenSSH_9.6pl1-Ubuntu-12.04.1
(general) compression: OpenSSH 8.4+, Dropbear SSH 2018.76+
(general) compression: enabled (zlib@openssh.com)

# Key exchange algorithms
(hex) ecdh-sha2-nistp256 -- [info] available since OpenSSH 8.0
(hex) ecdh-sha2-nistp384 -- [info] available since OpenSSH 7.4, Dropbear SSH 2018.76
(hex) curve25519-sha256@libssh.org -- [info] default key exchange since OpenSSH 6.4
(hex) curve25519-sha256@openssh.com -- [info] available since OpenSSH 6.4, Dropbear SSH 2013.62
(hex) default-key-exchange-nistp256 -- [info] default key exchange since OpenSSH 6.4
(hex) ecdh-sha2-nistp256 -- [fail] using elliptic curves that are suspected as being backdoored by the U.S. National Security Agency
(hex) ecdh-sha2-nistp384 -- [fail] using elliptic curves that are suspected as being backdoored by the U.S. National Security Agency
(hex) ecdh-sha2-nistp521 -- [fail] using elliptic curves that are suspected as being backdoored by the U.S. National Security Agency
(hex) diffie-hellman-group-exchange-sha256 (3072-bit) -- [info] available since OpenSSH 8.0
(hex) diffie-hellman-group16-sha512 -- [info] available since OpenSSH 7.1, Dropbear SSH 2016.73
(hex) diffie-hellman-group18-sha512 -- [info] available since OpenSSH 7.3
(hex) diffie-hellman-group14-sha256 -- [warn] 2048-bit modulus only provides 112-bits of symmetric strength
(hex) diffie-hellman-group18-sha512 -- [info] available since OpenSSH 7.3, Dropbear SSH 2016.73
(hex) ext-info-s -- [info] pseudo-algorithm that denotes the peer supports RFC3596 extensions
(hex) ext-strict-r-v00@openssh.com -- [info] pseudo-algorithm that denotes the peer supports a stricter key exchange method as a counter-measure to the Terrapin attack

# host-key algorithms
(hex) ecdsa-sha2-nistp256 -- [fail] using elliptic curves that are suspected as being backdoored by the U.S. National Security Agency
(hex) ecdsa-sha2-nistp384 -- [warn] using weak random number generator could reveal the key
(hex) ssh-ed25519 -- [info] available since OpenSSH 5.7, Dropbear SSH 2013.62
(hex) ssh-ed25519 -- [info] available since OpenSSH 6.5

# encryption algorithms (ciphers)
(enc) chacha20-poly1305@openssh.com -- [info] available since OpenSSH 6.5
(enc) aesi128-ctr -- [info] default cipher since OpenSSH 6.9
(enc) aesi192-ctr -- [info] available since OpenSSH 7.0, Dropbear SSH 0.52
(enc) aesi256-ctr -- [info] available since OpenSSH 7.1
(enc) aesi128-rc4@openssh.com -- [info] available since OpenSSH 6.2, Dropbear SSH 0.52
(enc) aesi256-rc4@openssh.com -- [info] available since OpenSSH 6.2

# message authentication code algorithms
(mac) umac-64-etm@openssh.com -- [warn] using small 64-bit tag size
(mac) umac-128-etm@openssh.com -- [info] available since OpenSSH 6.2
(mac) hmac-sha2-256-etm@openssh.com -- [info] available since OpenSSH 6.2
(mac) hmac-sha2-512-etm@openssh.com -- [info] available since OpenSSH 6.2
(mac) hmac-shal-etm@openssh.com -- [fail] using broken HMAC hash algorithm
(mac) umac-64@openssh.com -- [info] available since OpenSSH 6.2
(mac) umac-128@openssh.com -- [warn] using encrypt-and-MAC mode
(mac) hmac-sha2-256 -- [warn] using small 64-bit tag size
(mac) hmac-sha2-512 -- [info] available since OpenSSH 6.7
(mac) hmac-sha1 -- [info] available since OpenSSH 6.2
(mac) hmac-sha2-256 -- [info] available since OpenSSH 6.2
(mac) hmac-sha2-512 -- [info] available since OpenSSH 6.9, Dropbear SSH 2013.56
(mac) hmac-sha1 -- [info] available since OpenSSH 6.9, Dropbear SSH 2013.56
(mac) hmac-shal -- [info] available since OpenSSH 6.2
(mac) hmac-sha2-256 -- [warn] using encrypt-and-MAC mode
(mac) hmac-sha2-512 -- [info] available since OpenSSH 6.2
(mac) hmac-shal -- [info] available since OpenSSH 6.2

# fingerprints
(sha) ssh-ed25519: SHA256:8guUVFRD++vUDEmlhVSc2GejWUwG5UT9A21zNGfQyV'e

# algorithm recommendations (for OpenSSH 9.6)
(rec) -ecdh-sha2-nistp256 -- hex algorithm to remove
(rec) -ecdh-sha2-nistp384 -- hex algorithm to remove
(rec) -ecdh-sha2-nistp521 -- hex algorithm to remove
(rec) -curve25519-nistp256 -- hex algorithm to remove
(rec) -hmac-sha2-512 -- hex algorithm to remove
(rec) -hmac-sha2-256 -- hex algorithm to remove
(rec) -chacha20-poly1305@openssh.com -- hex algorithm to remove
(rec) +rsa-sha2-256 -- hex algorithm to append
(rec) +rsa-sha2-512 -- hex algorithm to append
(rec) +diffie-hellman-group14-sha256 -- hex algorithm to remove
(rec) -aes128-ctr -- hex algorithm to remove
(rec) -aes192-ctr -- hex algorithm to remove
(rec) -aes256-ctr -- hex algorithm to remove
(rec) -umac-128@openssh.com -- hex algorithm to remove
(rec) -umac-64-etm@openssh.com -- hex algorithm to remove
(rec) -umac-64@openssh.com -- hex algorithm to remove

# additional info
(info) For hardened guides on common OSes, please see: <https://www.ssh-audit.com/hardening_guides.html>
(info) Be aware that, while this target properly supports the strict key exchange method (via the Hex-strict-r-v00@openssh.com marker) needed to protect against the Terrapin vulnerability SSH channels with this target: chacha20-poly1305@openssh.com. If any CBC ciphers are in this list, you may remove them while leaving the *-etm@openssh.com MACs in place
```

Figura 7: ssh-audit inicial

- Mejoras aplicadas al *sshd_config*:

Tras analizar los resultados de la auditoría SSH, se implementaron las siguientes mejoras en el archivo de configuración */etc/ssh/sshd_config*:

```
#MEJORAR CONFIG#
KeyAlgorithms diffie-hellman-group-exchange-sha256,curve25519-sha256@libssh.org
HostKeyAlgorithms ssh-ed25519,rsa-sha2-256,rsa-sha2-512
Ciphers aes256-gcm@openssh.com,aes256-ctr
MACs hmac-sha2-256,hmac-sha2-512
```

Figura 8: Mejoras en el archivo *sshd_config*

Estas mejoras habilitaron algoritmos seguro de intercambio de claves, implementaron claves modernas, seleccionaron cifrados robusto y habilitaron algoritmos para proteger la integridad de los datos.

- Estado final de *sshd_config*:

Tras aplicar las mejoras, el resultado de la auditoría SSH muestra todo en verde, lo que confirma que el servidor es seguro y robusto.

```
azureuser@vm:~$ sudo ssh-audit 20.199.93.120
# general
(gen) banner: SSH-2.0-OpenSSH_9.6p1 Ubuntu-3ubuntu13.5
(gen) software: OpenSSH 9.6p1
(gen) compatibility: OpenSSH 7.4+, Dropbear SSH 2018.76+
(gen) compression: enabled (zlib@openssh.com)

# key exchange algorithms
(kex) curve25519-sha256@libssh.org      -- [info] available since OpenSSH 6.4, Dropbear SSH 2013.62
                                              '- [info] default key exchange since OpenSSH 6.4
(kex) curve25519-sha256                  -- [info] available since OpenSSH 7.4, Dropbear SSH 2018.76
                                              '- [info] default key exchange since OpenSSH 6.4
(kex) diffie-hellman-group16-sha512       -- [info] available since OpenSSH 7.3, Dropbear SSH 2016.73
(kex) diffie-hellman-group18-sha512       -- [info] available since OpenSSH 7.3
(kex) diffie-hellman-group-exchange-sha256 (3072-bit) -- [info] available since OpenSSH 4.4
                                              '- [info] OpenSSH's GEX fallback mechanism was triggered during testing. Very old SSH
o disabled by recompiling the code (see https://github.com/openssl/openssl-portable/blob/V_9_4_dh.c#L477).
(kex) ext-info-s                         -- [info] pseudo-algorithm that denotes the peer supports RFC8308 extensions
(kex) kex-strict-s-v00@openssh.com        -- [info] pseudo-algorithm that denotes the peer supports a stricter key exchange method as a co

# host-key algorithms
(key) ssh-ed25519                      -- [info] available since OpenSSH 6.5

# encryption algorithms (ciphers)
(enc) aes256-gcm@openssh.com            -- [info] available since OpenSSH 6.2
(enc) aes256-ctr                        -- [info] available since OpenSSH 3.7, Dropbear SSH 0.52
(enc) chacha20-poly1305@openssh.com     -- [info] available since OpenSSH 6.5
                                              '- [info] default cipher since OpenSSH 6.9

# message authentication code algorithms
(mac) hmac-sha2-256-otm@openssh.com    -- [info] available since OpenSSH 6.2
(mac) hmac-sha2-512-otm@openssh.com    -- [info] available since OpenSSH 6.2

# fingerprints
(fin) ssh-ed25519: SHA256:8gUVfRO+vUOEwlhVSc2GaQMU4zG5UT9A2lzNGfqvYe

# algorithm recommendations (for OpenSSH 9.6)
(rec) +aes128-ctr                      -- enc algorithm to append
(rec) +aes128-gcm@openssh.com           -- enc algorithm to append
(rec) +aes192-ctr                      -- enc algorithm to append
(rec) +rsa-sha2-256                     -- key algorithm to append
(rec) +rsa-sha2-512                     -- key algorithm to append
(rec) +sntrup761x25519-sha512@openssh.com -- kex algorithm to append

# additional info
(INFO) Be aware that, while this target properly supports the strict key exchange method (via the kex-strict-?-v00@openssh.com marker) needed
e vulnerability will still be present. The following algorithms would allow an unpatched peer to create vulnerable SSH channels with this t
.com MACs in place; these MACs are fine while paired with non-CBC cipher types.
```

Figura 9: ssh-audit final

3.1.6. Protección contra ataques de fuerza bruta con Fail2ban

Para mitigar ataques de fuerza bruta en el servidor SSH, se ha configurado el *jail sshd* específico en Fail2Ban. Este *jail* supervisa los intentos de inicio de sesión registrados en los logs del sistema y aplica un bloqueo automático de 10 minutos a las direcciones IP que superan el umbral de 3 intentos fallidos en un período de 10 minutos.

```
[sshd]
enabled = true
port      = ssh
filter    = sshd
logpath   = /var/log/auth.log
maxretry  = 3
bantime  = 600
findtime  = 600
action    = iptables[name=SSH, port=ssh, protocol=tcp]
```

Figura 10: *jail [sshd]*

Para comprobar su correcto funcionamiento, intentamos acceder por SSH a nuestra VM desde otro dispositivo usando un **usuario incorrecto**:

```
C:\Users\berna>ssh user@20.199.93.120
The authenticity of host '20.199.93.120 (20.199.93.120)' can't be established.
ED25519 key fingerprint is SHA256:8gUVfR0++vUOEwLhVSc2GaQWU4zG5UT9A21zNGfqvYo.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '20.199.93.120' (ED25519) to the list of known hosts.
user@20.199.93.120: Permission denied (publickey).

C:\Users\berna>ssh user@20.199.93.120
user@20.199.93.120: Permission denied (publickey).

C:\Users\berna>ssh user@20.199.93.120
user@20.199.93.120: Permission denied (publickey).

C:\Users\berna>
```

A los 3 intentos de conexión, podemos comprobar que se ha baneado la IP del dispositivo en el que estábamos realizando las pruebas, viendo el estado del *jail*:

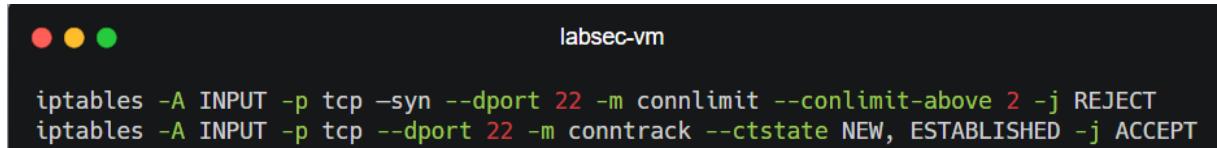
```
azureuser@vm:~$ sudo fail2ban-client status sshd
Status for the jail: sshd
|- Filter
| |- Currently failed: 2
| |- Total failed:     10
| '- File list:          /var/log/auth.log
`- Actions
  |- Currently banned: 1
  |- Total banned:     2
  '- Banned IP list:    100.100.100.100
```

El filtro al que hacemos referencia ya está configurado y se encuentra en el directorio */etc/fail2ban/filter.d/*.

3.1.7. Limitación del número de sesiones SSH con iptables

Como medida adicional de seguridad, configuraremos el servidor SSH para permitir un máximo de 2 sesiones simultáneas por usuario.

Para ello, usaremos **iptables**, una herramienta que define las reglas de un firewall para filtrar el tráfico mediante **Netfilter**, y agregaremos las siguientes reglas:



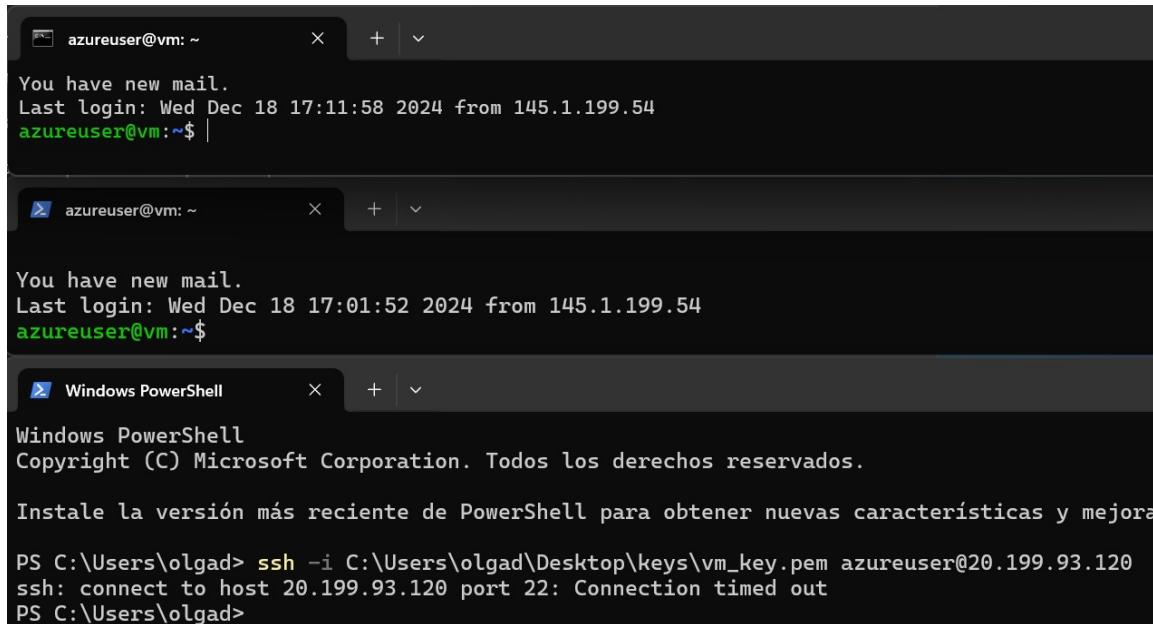
```
labsec-vm

iptables -A INPUT -p tcp --syn --dport 22 -m connlimit --connlimit-above 2 -j REJECT
iptables -A INPUT -p tcp --dport 22 -m conntrack --ctstate NEW, ESTABLISHED -j ACCEPT
```

Figura 11: Configuración para limitar el número de sesiones

- La primera regla limita las conexiones SSH entrantes para cada dirección IP a un máximo de 2. Si una IP intenta abrir más de 2 conexiones SSH simultáneas, las conexiones adicionales serán rechazadas.
- La segunda regla asegura que las conexiones SSH nuevas y las ya establecidas (que no están bloqueadas por la primera regla) sean aceptadas.

Para comprobar el correcto funcionamiento, hemos intentado abrir 3 sesiones. Las 2 primeras han sido exitosas pero la tercera ya no, apareciendo un mensaje que indica que se ha producido un time out.



```
azureuser@vm: ~
You have new mail.
Last login: Wed Dec 18 17:11:58 2024 from 145.1.199.54
azureuser@vm:~$ |
```



```
azureuser@vm: ~
You have new mail.
Last login: Wed Dec 18 17:01:52 2024 from 145.1.199.54
azureuser@vm:~$ |
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras.

PS C:\Users\olgad> ssh -i C:\Users\olgad\Desktop\keys\vm_key.pem azureuser@20.199.93.120
ssh: connect to host 20.199.93.120 port 22: Connection timed out
PS C:\Users\olgad>
```

Figura 12: Intento de conexión de 3 sesiones simultáneas

La configuración de iptables se ha hecho persistente para garantizar que las reglas se mantengan activas incluso después de reiniciar el sistema.

3.2. Configuración de Nginx

3.2.1. Instalación y configuración básica

Para empezar con la configuración de Nginx primero se instaló la versión *1.24.0* del servicio. Antes de realizar cualquier modificación, se creó una copia de seguridad del **archivo de configuración principal**, *nginx.conf*, ubicado en */etc/nginx/*, con el fin de preservar la configuración original en caso de ser necesario restaurarla.

Posteriormente, se creó un nuevo archivo de configuración llamado *labsec-vm.francecentral.cloudapp.azure.com.conf*, en la ubicación */etc/nginx/sites-available/*. Este archivo define la configuración específica para el **servidor virtual** asociado al dominio proporcionado por nuestra máquina virtual en Azure, *labsec-vm.francecentral.cloudapp.azure.com*.

Para activar esta configuración, se generó un enlace simbólico del archivo en la carpeta */etc/nginx/sites-enabled/*, lo que permite que Nginx cargue automáticamente esta configuración al reiniciarse. Esta estructura de directorios separa los archivos disponibles de los activos, facilitando la gestión de configuraciones múltiples.

Las modificaciones realizadas en el archivo de configuración principal y en de la configuración del servidor virtual se explican en detalle en los apartados siguientes.

3.2.2. Implementación de CertBot

Para asegurar las comunicaciones entre los usuarios y nuestro servidor web, hemos implementado un certificado SSL/TLS gratuito utilizando **Certbot** con la autoridad de certificación **Let's Encrypt**. Esto garantiza que todas las conexiones al sitio web se realicen de manera segura y cifrada.

```
#CertBot#
listen [:]:443 ssl ipv6only=on; # managed by Certbot
listen 443 ssl; # managed by Certbot
ssl_certificate /etc/letsencrypt/live/labsec-vm.francecentral.cloudapp.azure.com/fullchain.pem; # managed by Certbot
ssl_certificate_key /etc/letsencrypt/live/labsec-vm.francecentral.cloudapp.azure.com/privkey.pem; # managed by Certbot
include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
```

Figura 13: Certificado y clave privada generados

Certbot también configuró la **redirección de tráfico HTTP (puerto 80) a HTTPS (puerto 443)**. Esto garantiza que las conexiones siempre sean cifradas.

```
server {
    if ($host = labsec-vm.francecentral.cloudapp.azure.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
    listen [:]:80;

    server_name labsec-vm.francecentral.cloudapp.azure.com;
    return 404; # managed by Certbot

}
```

Figura 14: Regla de redirección de tráfico HTTP a HTTPS

```
azureuser@vm:~$ wget http://labsec-vm.francecentral.cloudapp.azure.com
--2024-12-18 17:16:38--  http://labsec-vm.francecentral.cloudapp.azure.com/
Resolving labsec-vm.francecentral.cloudapp.azure.com (labsec-vm.francecentral.cloudapp.azure.com)... 20.199.93.120
Connecting to labsec-vm.francecentral.cloudapp.azure.com (labsec-vm.francecentral.cloudapp.azure.com)|20.199.93.120|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://labsec-vm.francecentral.cloudapp.azure.com/ [following]
--2024-12-18 17:16:38--  https://labsec-vm.francecentral.cloudapp.azure.com/
Connecting to labsec-vm.francecentral.cloudapp.azure.com (labsec-vm.francecentral.cloudapp.azure.com)|20.199.93.120|:443... connected.
HTTP request sent, awaiting response... 200 OK
```

Figura 15: Prueba de redirección de tráfico HTTP a HTTPS

Como podemos ver en las 2 figuras anteriores, cuando un usuario intenta acceder a un sitio web utilizando el puerto 80 o escribiendo HTTP en la URL (*http://labsec-vm.francecentral.cloudapp.azure.com*), se genera una redirección hacia el puerto 443, el predeterminado para HTTPS. Esta redirección se realiza mediante una respuesta HTTP **301 Moved Permanently**.

Finalmente podemos comprobar que al acceder a la página web nos aparece que la conexión es segura y podemos ver el certificado que lo asegura.

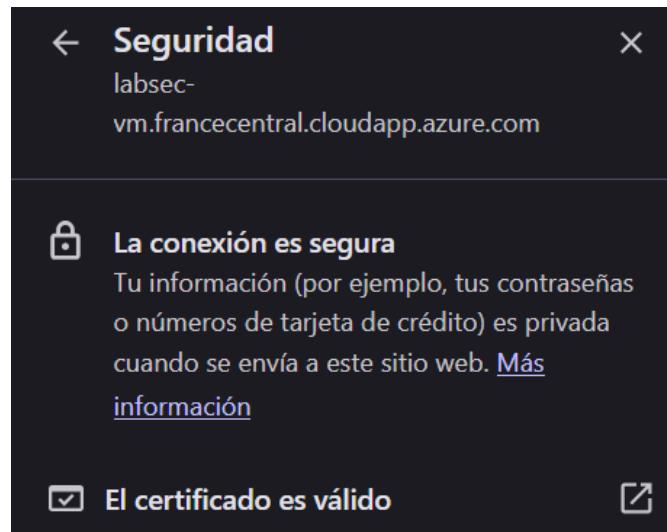


Figura 16: Mensaje de conexión segura

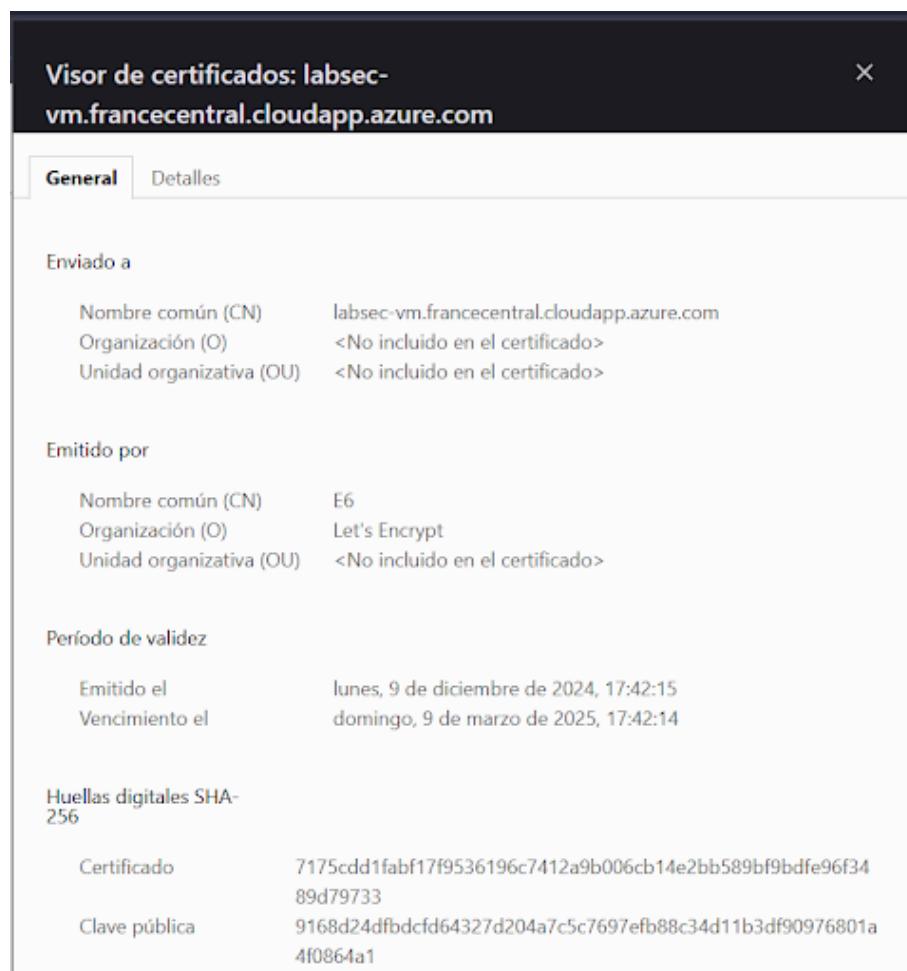


Figura 17: Certificado SSL/TLS

3.2.3. Archivo de configuración principal

En este apartado se explicarán las modificaciones que se han realizado en el archivo `nginx.conf`. Este fichero tiene la función de ser la base central de configuración para el servidor web, define directivas globales y ajustes que afectan el comportamiento de **Nginx**, además de incluir configuraciones específicas de otros ficheros.

A continuación, procedemos a explicar las directivas adicionales aplicadas para mejorar la seguridad de nuestro servicio, partiendo de la configuración que viene establecida por defecto.

En primer lugar, establecemos la directiva `load_module`, utilizada para cargar módulos externos en tiempo de ejecución. En este caso, el de **ModSecurity**, explicado en detalle más adelante.

Dentro de la configuración del bloque `http`, encargado de configurar todas las funciones HTTP que usa Nginx, hemos realizado las siguientes modificaciones:

I. `tcp_nodelay on`:

Envía paquetes sin retrasos, útil en conexiones pequeñas.

II. `keepalive_timeout 65`:

Definimos 65 segundos como el tiempo que una conexión HTTP puede permanecer abierta en espera de más solicitudes.

III. `server_tokens off`:

Deshabilitar los tokens del servidor hace que sea más difícil determinar la versión de Nginx y, por lo tanto, más difícil para un atacante ejecutar ataques específicos de la versión.

IV. `ssl_protocols TLSv1.2 TLSv1.3`:

Únicamente permite las versiones más modernas y por tanto seguras del protocolo SSL (TLS 1.2 y TLS 1.3).

V. `proxy_hide_header X-Powered-By`:

Esta directiva se utiliza para ocultar los encabezados de la respuesta de un servidor proxy a un cliente. Concretamente, oculta las cabeceras de tipo `X-Powered-By`.

VI. `add_header X-Frame-Options SAMEORIGIN`:

Se utiliza para evitar que otras páginas muestren el sitio web dentro de un marco y engañen a los usuarios para que hagan clic o realicen acciones sin saberlo.

VII. `limit_req_zone $binary_remote_addr zone=req_limit_per_ip:10m rate=1r/s;;`

Sirve para limitar la tasa de peticiones de los usuarios, necesario para la configuración de *Fail2ban*, explicado más adelante.

Con estas nuevas directivas aplicadas, podemos comprobar la información referente a la versión de Nginx que aparece al acceder a ciertos sitios como por ejemplo uno no permitido ahora se encuentra ocultada, y ver como antes no era así:

403 Forbidden

nginx/1.24.0 (Ubuntu)

Figura 18: *Server Header* antes de aplicar las directivas

403 Forbidden

nginx

Figura 19: *Server Header* después de aplicar las directivas

Lo mismo a la hora de usar la función *Inspeccionar* de *Google Chrome* y ver las cabeceras de las peticiones HTTP que se realizan, en concreto la de *Server*:

Request Method:	GET
Status Code:	200 OK
Remote Address:	20.199.93.120:443
Referrer Policy:	strict-origin-when-cross-origin
▼ Response Headers	<input type="checkbox"/> Raw
Connection:	keep-alive
Content-Encoding:	gzip
Content-Type:	text/html
Date:	Wed, 18 Dec 2024 17:38:14 GMT
Etag:	W/"6759aad7-62a"
Last-Modified:	Wed, 11 Dec 2024 15:08:07 GMT
Server:	nginx

Finalmente el archivo de configuración queda de la siguiente forma:

```
● ● ● /etc/nginx/nginx.conf

user www-data;
worker_processes auto;
pid /run/nginx.pid;
error_log /var/log/nginx/error.log;
include /etc/nginx/modules-enabled/*.conf;
load_module /etc/nginx/modules/ngx_http_modsecurity_module.so;

events {
    worker_connections 768;
}

http {

    ##
    # Basic Settings
    ##

    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;
    server_tokens off;

    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    ##
    # SSL Settings
    ##

    #Solo permitir las versiones más modernas
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;

    ##
    # Logging Settings
    ##

    access_log /var/log/nginx/access.log;

    ##
    # Gzip Settings
    ##

    gzip on;

    #Configuración adicional
    proxy_hide_header X-Powered-By;
    add_header X-Frame-Options SAMEORIGIN;

    #Configuración para fail2ban nginx-limit-req
    limit_req_zone $binary_remote_addr zone=req_limit_per_ip:10m rate=1r/s;

    ##
    # Virtual Host Configs
    ##

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```

Figura 20: Archivo de configuración principal *nginx.conf*

3.2.4. Archivo de configuración del servidor virtual labsec-vm

En este apartado se explicarán las directivas aplicadas en el fichero *labsec-vm.francecentral.cloudapp.azure.com.conf*, cuya función es la de configurar los ajustes específicos para el sitio web que se aloja en nuestro servidor.

En primer lugar, ajustamos el *server_name* con el nombre de dominio que nos proporciona Azure (*labsec-vm.francecentral.cloudapp.azure.com*) e indicamos el directorio donde se encuentra el código html de nuestro sitio web (*/var/www/labsec-vm.francecentral.cloudapp.azure.com*).

Ahora bien, las principales directivas del archivo se definen a continuación:

I. *ModSecurity*:

Se activa el módulo *ModSecurity*, explicado posteriormente, y se define la ubicación del archivo de reglas (*main.conf*) de dicho módulo.

II. A continuación se añaden tres bloques de *locations*. En cada bloque se define cómo manejar las solicitudes a una ruta específica:

- *location ~ / \ . :*

Tiene la función de proteger archivos ocultos o sensibles en un servidor web basado en Nginx, denegando totalmente el acceso.

- *location /admin:*

Restringe el acceso a la página */admin*, de manera que solo puede acceder la IP pública específica *81.203.19.45*. Si cualquier otra IP intenta acceder, se mostrará el mensaje de error **403 Forbidden**.

- *location /private/:*

Se define una ubicación para */private/*, que está protegida por una autenticación basada en usuario y contraseña, en la cual solo si se conocen las credenciales se podrá entrar.

Las pruebas del correcto funcionamiento de las 3 *locations* se realizarán en el siguiente apartado llamado *Locations*.

III. *Gzip*:

Activamos la compresión *Gzip*, que reduce el tamaño de los archivos transferidos y mejora la velocidad de carga. Concretamente, definimos el nivel 3 de compresión, y se especifican los tipos de archivos que deben ser comprimidos.

IV. Headers:

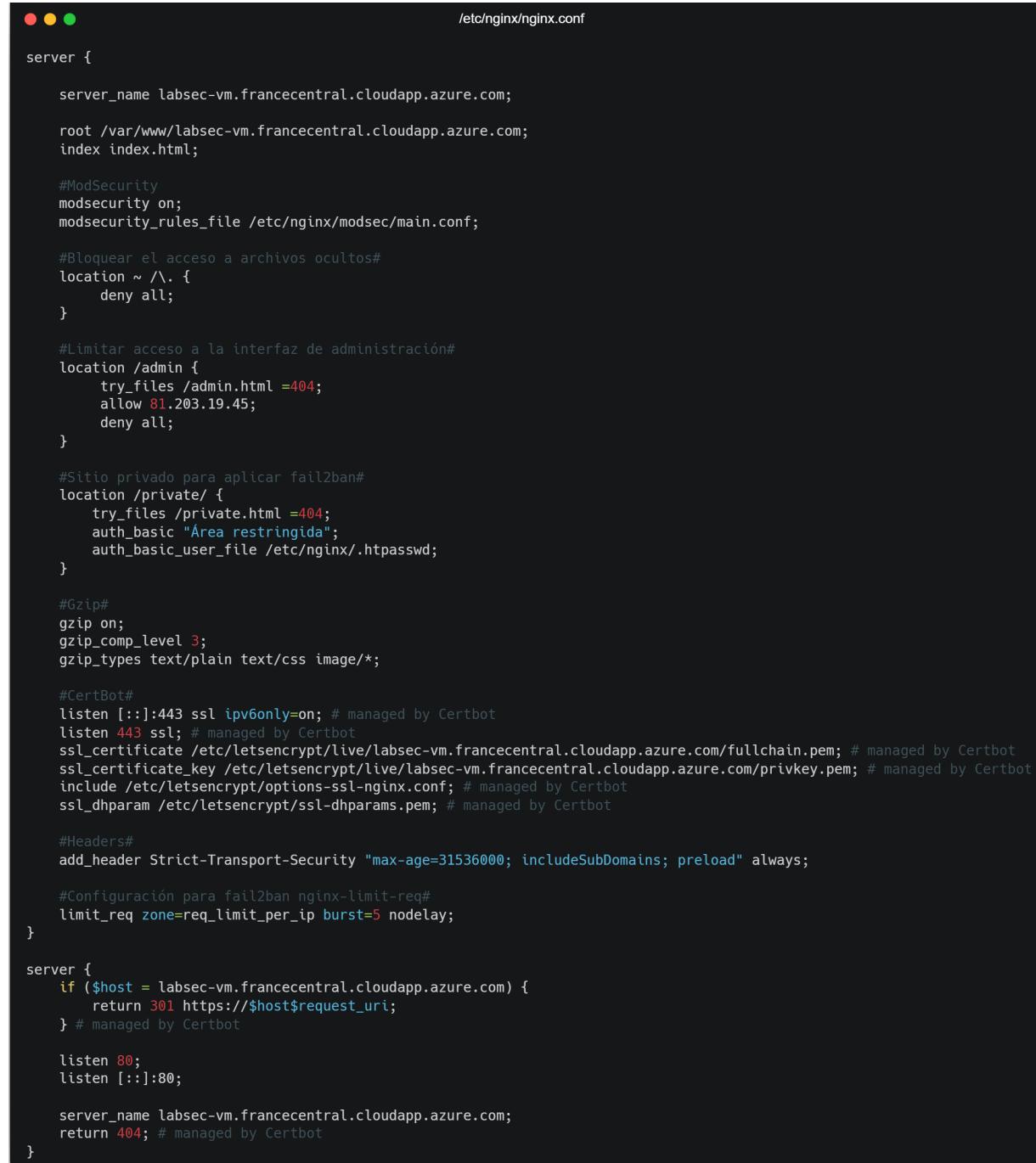
Activa HSTS (HTTP Strict Transport Security), un mecanismo de seguridad que obliga a los navegadores a comunicarse con el servidor web solo a través de HTTPS durante un periodo determinado. Esta configuración mejora la seguridad contra ataques como el *downgrade de protocolo* o el *man-in-the-middle*.

v. Configuración para el *Jail* de *Fail2ban* [*nginx-limit-req*]:

Configuramos un límite en la cantidad de solicitudes que un usuario puede hacer en un período de tiempo. Concretamente se permiten hasta 5 solicitudes.

La configuración realizada automáticamente por **Certbot** ya se encuentra explicada en el anterior apartado *Implementación de Certbot*.

Finalmente el archivo de configuración queda de la siguiente forma:



```
/etc/nginx/nginx.conf

server {
    server_name labsec-vm.francecentral.cloudapp.azure.com;
    root /var/www/labsec-vm.francecentral.cloudapp.azure.com;
    index index.html;

    #ModSecurity
    modsecurity on;
    modsecurity_rules_file /etc/nginx/modsec/main.conf;

    #Bloquear el acceso a archivos ocultos#
    location ~ /\. {
        deny all;
    }

    #Limitar acceso a la interfaz de administración#
    location /admin {
        try_files /admin.html =404;
        allow 81.203.19.45;
        deny all;
    }

    #Sitio privado para aplicar fail2ban#
    location /private/ {
        try_files /private.html =404;
        auth_basic "Área restringida";
        auth_basic_user_file /etc/nginx/.htpasswd;
    }

    #Gzip#
    gzip on;
    gzip_comp_level 3;
    gzip_types text/plain text/css image/*;

    #CertBot#
    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/labsec-vm.francecentral.cloudapp.azure.com/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/labsec-vm.francecentral.cloudapp.azure.com/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

    #Headers#
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains; preload" always;

    #Configuración para fail2ban nginx-limit-req#
    limit_req zone=req_limit_per_ip burst=5 nodelay;
}

server {
    if ($host = labsec-vm.francecentral.cloudapp.azure.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
    listen [::]:80;

    server_name labsec-vm.francecentral.cloudapp.azure.com;
    return 404; # managed by Certbot
}
```

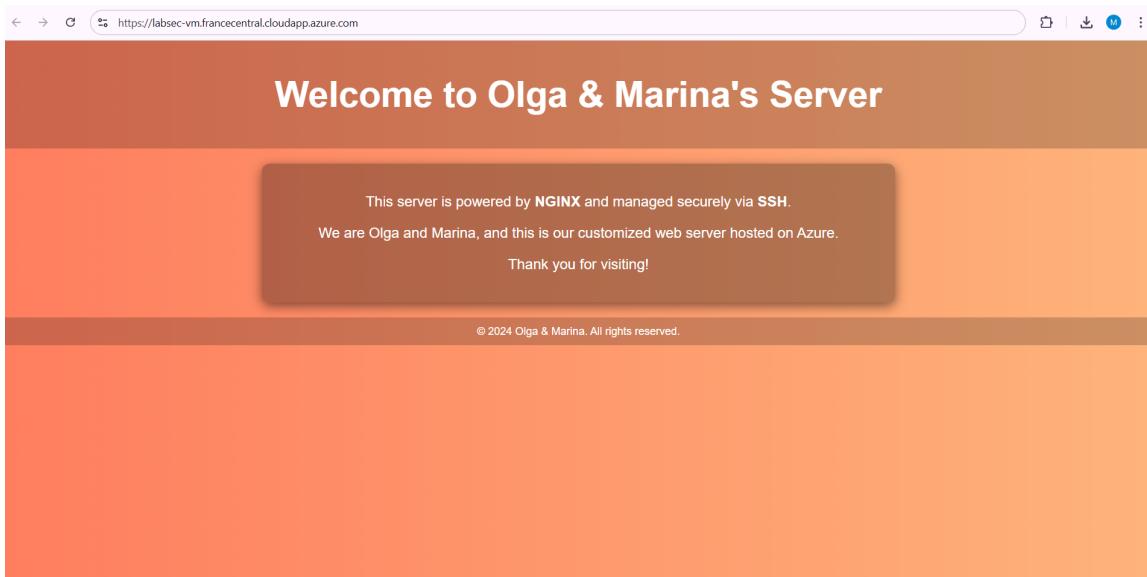
Figura 21: Archivo de configuración del servidor virtual *labsec-vm.conf*

3.2.5. Locations

En esta sección, haremos las búsquedas en el navegador necesarias para comprobar que las *locations* configuradas y mostradas anteriormente funcionan de forma correcta.

- I. <https://labsec-vm.francecentral.cloudapp.azure.com>

El resultado que se muestra al realizar la búsqueda en el navegador tanto mediante HTTPS como HTTP es el siguiente:



- II. <https://labsec-vm.francecentral.cloudapp.azure.com/admin>

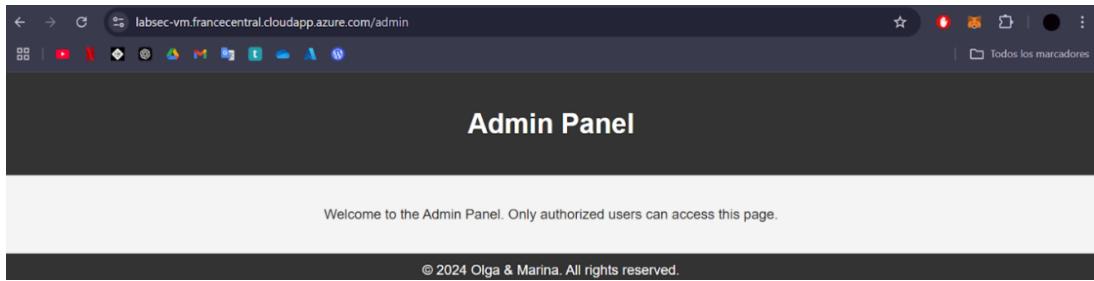
Si intentamos acceder a este dominio con una dirección IP diferente a la especificada, nos encontramos el error **403 Forbidden**:



También podemos comprobar que el acceso es denegado con la herramienta *curl*:

```
PS C:\Users\olgad> curl http://labsec-vm.francecentral.cloudapp.azure.com/admin
curl : 403 Forbidden
nginx/1.24.0 (Ubuntu)
En línea: 1 Carácter: 1
+ curl http://labsec-vm.francecentral.cloudapp.azure.com/admin
+ ~~~~~
+ CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebException
+ FullyQualifiedErrorMessage : WebCmdletWebResponseException, Microsoft.PowerShell.Commands.InvokeWebRequestCommand
```

En cambio, si accedemos desde la IP específica, sí que podremos ver el contenido:



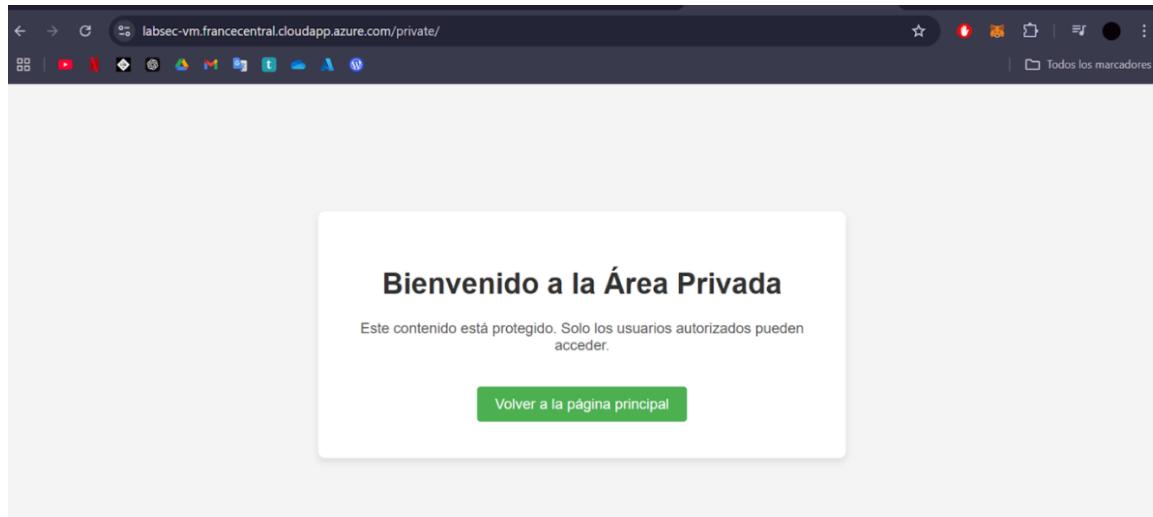
Para acabar, volvemos a comprobar que esta vez sí que nos deja acceder al dominio usando *curl*.

```
PS C:\Users\olgad> curl http://labsec-vm.francecentral.cloudapp.azure.com/admin

StatusCode      : 200
StatusDescription : OK
Content         : <!DOCTYPE html>
                  <html lang="en">
                  <head>
                      <meta charset="UTF-8">
```

III. *https://labsec-vm.francecentral.cloudapp.azure.com/private/*

Para poder acceder, necesitamos escribir el usuario y contraseña correctos (**usuario** y **usuario** en este caso), y podremos ver el contenido con total normalidad:



3.2.6. Protección con Fail2ban

En esta sección haremos uso de la herramienta *Fail2ban*, la cual nos servirá para evitar accesos no autorizados al servidor. Funciona mediante *jails*, que son configuraciones usadas para monitorizar registros específicos y ejecutar acciones de protección.

Una vez instalada la herramienta, podremos ver que su configuración se encuentra en `/etc/fail2ban/`. Allí, encontraremos un archivo llamado `jail.conf` con las configuraciones del sistema. El primer paso será hacer una copia de dicho archivo para tenerlo como backup y así poder modificar el nuevo, llamado `jail.local`.

Dentro de `jail.local` habilitaremos los *jails* que consideremos. A continuación, se detallan los que hemos configurado, juntamente con las pruebas para comprobar que funcionan de la manera correcta:

I. [nginx-http-auth]

Este *jail* se utiliza para bloquear direcciones IP que fallen a la hora de autenticarse (en HTTP y HTTPS) en un tiempo determinado.

```
[nginx-http-auth]
# mode = normal
enabled = true
port    = http,https
filter   = nginx-http-auth
logpath = /var/log/nginx/error.log
maxretry= 5
bantime = 600
#logpath = %(nginx_error_log)s
```

Figura 22: [nginx-http-auth]

La opción `enabled` establecida en `true` se utiliza para habilitar el filtro. Especificamos los puertos a los cuales aplicamos el filtro, es decir, HTTP y HTTPS. Además, en nuestro caso, vamos a permitir un máximo de 5 intentos antes de bloquear una dirección IP. En caso de que una IP exceda estos 5 intentos, será bloqueada durante 10 minutos.

Comprobación:

A continuación, para comprobar que el *jail* funciona correctamente, accedemos a `https://labsec-vm.francecentral.cloudapp.azure.com/private/` en nuestro navegador. Nginx solicita un nombre de usuario y contraseña.

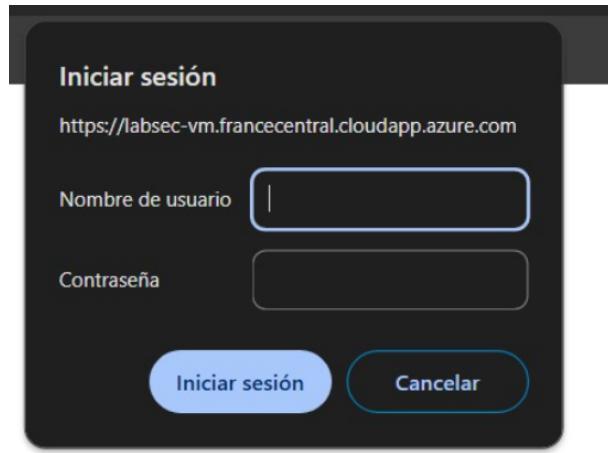


Figura 23: Solicitud de autenticación para la *location /private/*

Si escribimos más de 5 veces la contraseña o el usuario de manera incorrecta, nos bloqueará. Podemos comprobarlo con este comando:

```
azureuser@vm:~$ sudo fail2ban-client status nginx-http-auth
Status for the jail: nginx-http-auth
|- Filter
|  |- Currently failed: 0
|  |- Total failed:      6
|  '- File list:          /var/log/nginx/error.log
`- Actions
  |- Currently banned: 1
  |- Total banned:     1
  '- Banned IP list:    100.000.00.0
```

También podemos llegar a ver desde los logs los intentos realizados por el usuario antes de ser baneado.

```
2024/12/13 12:43:56 [error] 5199#5199: *11 user "dsd" was not fo
und in "/etc/nginx/.htpasswd", client: ..., server: lab
sec-vm.francecentral.cloudapp.azure.com, request: "GET /private/
HTTP/1.1", host: "labsec-vm.francecentral.cloudapp.azure.com"
2024/12/13 12:44:13 [error] 5199#5199: *13 user "user" was not f
ound in "/etc/nginx/.htpasswd", client: ..., server: la
bsec-vm.francecentral.cloudapp.azure.com, request: "GET /private
/ HTTP/1.1", host: "labsec-vm.francecentral.cloudapp.azure.com"
2024/12/13 12:44:16 [error] 5266#5266: *1 user "admin" was not f
ound in "/etc/nginx/.htpasswd", client: ..., server: la
bsec-vm.francecentral.cloudapp.azure.com, request: "GET /private
/ HTTP/1.1", host: "labsec-vm.francecentral.cloudapp.azure.com"
2024/12/13 12:44:21 [error] 5266#5266: *1 user "admin1" was not
found in "/etc/nginx/.htpasswd", client: ..., server: l
absec-vm.francecentral.cloudapp.azure.com, request: "GET /privat
e/ HTTP/1.1", host: "labsec-vm.francecentral.cloudapp.azure.com"
2024/12/13 12:44:29 [error] 5266#5266: *1 user "user" was not fo
und in "/etc/nginx/.htpasswd", client: ..., server: lab
sec-vm.francecentral.cloudapp.azure.com, request: "GET /private/
HTTP/1.1", host: "labsec-vm.francecentral.cloudapp.azure.com"
```

Figura 24: Intentos de usuario y contraseña fallidos

II. [nginx-limit-req]

Esta sección limita la cantidad de solicitudes que un usuario puede realizar en un tiempo determinado.

```
[nginx-limit-req]
enabled = true
port     = http,https
filter   = nginx-limit-req
logpath  = /var/log/nginx/error.log
maxretry= 10
bantime = 900
#logpath = %(nginx_error_logs)
```

Figura 25: [nginx-limit-req]

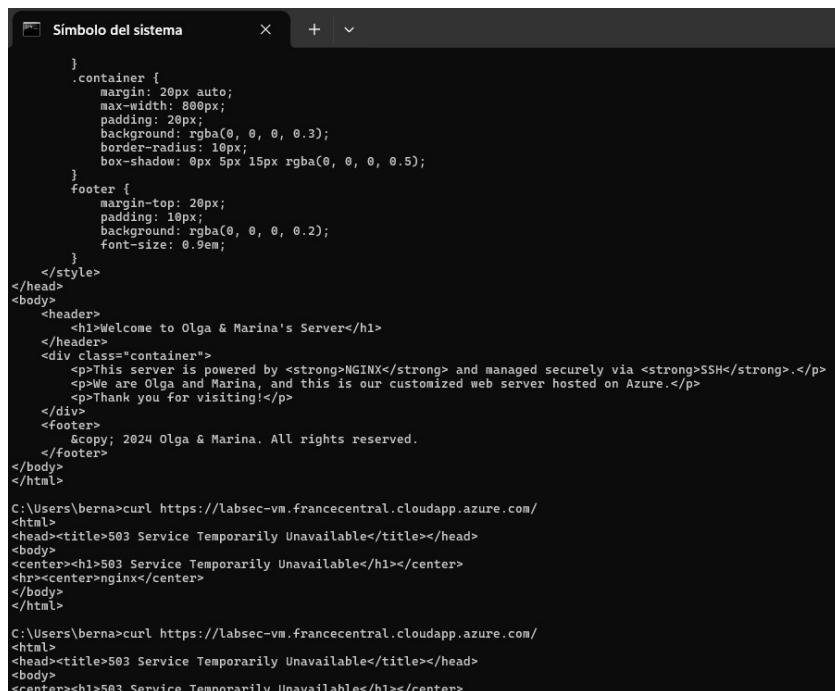
De la misma manera que antes, habilitamos el filtro y especificamos los puertos. El número máximo de intentos de solicitud permitidos antes de bloquear una IP será de 10. El tiempo durante el cual una IP será bloqueada es de 15 minutos.

Comprobación:

Podemos comprobar que este *jail* esté funcionando correctamente si realizamos múltiples solicitudes al servidor web, superando el límite de solicitudes configurado previamente. Así pues, después de realizar dicha cantidad, comprobamos el registro *error.log*, para observar que se han detectado las solicitudes a nuestro dominio:

```
r: labsec-vm.francecentral.cloudapp.azure.com, request: "GET / HTTP/1.1", host: "labsec-vm.francecentral.cloudapp.azure.com"
2024/12/16 18:28:33 [error] 10802#10802: *16 limiting requests, excess: 5.360 by zone "req_limit_per_ip", client: -----, server: labsec-vm.francecentral.cloudapp.azure.com, request: "GET / HTTP/1.1", host: "labsec-vm.francecentral.cloudapp.azure.com"
2024/12/16 18:28:33 [error] 10802#10802: *17 limiting requests, excess: 5.188 by zone "req_limit_per_ip", client: -----, server: labsec-vm.francecentral.cloudapp.azure.com, request: "GET / HTTP/1.1", host: "labsec-vm.francecentral.cloudapp.azure.com"
```

Al decimoprimer intento, al atacante le aparecerá un mensaje de **503 Service Temporarily Unavailable** como vemos a continuación y su IP será baneada.



III. [nginx-botsearch]

Este *jail* sirve para detectar y bloquear los bots maliciosos que intenten realizar solicitudes no válidas o repetitivas en el servidor Nginx.

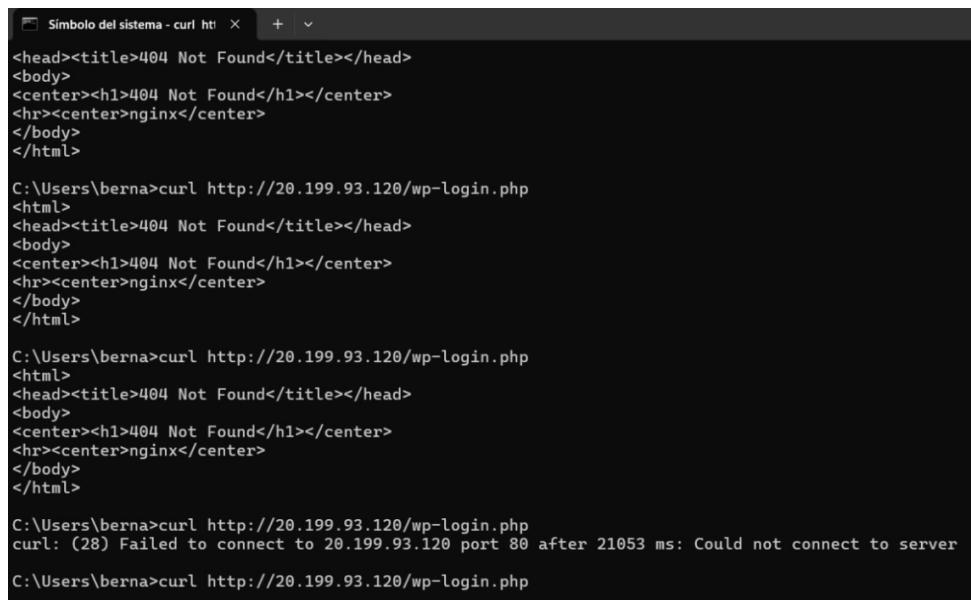
```
[nginx-botsearch]
enabled = true
port    = http,https
filter   = nginx-botsearch
logpath = /var/log/nginx/access.log
maxretry= 5
bantime = 3600
#logpath = %(nginx_error_log)s
```

Figura 26: [nginx-botsearch]

Como podemos observar, el número máximo de intentos fallidos permitidos antes de bloquear una IP es de 5. El tiempo durante el cual dicha IP será bloqueada es de 1 hora.

Comprobación:

Para comprobar el funcionamiento de este último *jail* se debe simular el comportamiento de un bot malicioso que realiza intentos de acceso a recursos sensibles. Mediante la herramienta *curl*, vamos a intentar acceder repetidamente a rutas del servidor Nginx que sean comúnmente atacadas por bots. En nuestro caso, vamos a intentar acceder a un recurso específico de WordPress: */wp-login.php*.



A screenshot of a terminal window titled "Símbolo del sistema - curl http". The window displays three consecutive curl commands being run from the directory "C:\Users\berna". Each command attempts to connect to the URL "http://20.199.93.120/wp-login.php". The first two attempts result in a 404 Not Found error, while the third attempt fails with the message "curl: (28) Failed to connect to 20.199.93.120 port 80 after 21053 ms: Could not connect to server".

```
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>

C:\Users\berna>curl http://20.199.93.120/wp-login.php
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>

C:\Users\berna>curl http://20.199.93.120/wp-login.php
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>

C:\Users\berna>curl http://20.199.93.120/wp-login.php
curl: (28) Failed to connect to 20.199.93.120 port 80 after 21053 ms: Could not connect to server
C:\Users\berna>curl http://20.199.93.120/wp-login.php
```

Como vemos, al sexto intento aparece un mensaje que indica que ya no puede conectarse al servidor.

Tambien podemos observar como *Fail2Ban* bloquea la IP que realiza esos intentos:

```
azureuser@vm:~$ sudo fail2ban-client status nginx-botsearch
Status for the jail: nginx-botsearch
|- Filter
|  |- Currently failed: 0
|  |- Total failed:      5
|  |- File list:          /var/log/nginx/access.log
`- Actions
   |- Currently banned: 1
   |- Total banned:     1
   `- Banned IP list:    10.0.0.100
```

Los filtros a los que hacemos referencia se encuentran en el directorio */etc/fail2ban/filter.d/*. Estos filtros ya se encuentran configurados y no hace falta modificarlos.

Estos son los *jails* activos en nuestro servicio:

```
azureuser@vm:~$ sudo fail2ban-client status
Status
|- Number of jail:      4
`- Jail list:    nginx-botsearch, nginx-http-auth, nginx-limit-re
q, sshd
```

3.2.7. Firewall ModSecurity con reglas OWASP

A continuación, pasaremos a explicar cómo configurar adecuadamente **ModSecurity** para Nginx. Es un firewall de aplicaciones web (WAF) que ayuda a proteger nuestra web contra ataques comunes.

Debido a que ModSecurity no está oficialmente soportado como un módulo de Nginx, necesitaremos utilizar el conector *ModSecurity-nginx*. Dicho conector proporciona un canal de comunicación entre Nginx y *ModSecurity*.

Editaremos el archivo *nginx.conf* para así cargar el módulo de ModSecurity, como hemos explicado con anterioridad:

```
load_module /etc/nginx/modules/ngx_http_modsecurity_module.so;
```

Figura 27: Carga del módulo *ModSecurity*

Una vez instalado y configurado ModSecurity, el siguiente paso es integrar y activar el conjunto de reglas OWASP Core Rule Set (CRS), que ofrece una protección genérica contra ataques web comunes como inyecciones SQL, cross-site scripting (XSS), y otros riesgos identificados por OWASP.

El proceso comienza eliminando el conjunto de reglas predeterminado que viene preempaquetado con ModSecurity, ya que estas reglas están pensadas solo para pruebas iniciales. Después, se clona el repositorio oficial de OWASP-CRS desde GitHub en el directorio */usr/share/modsecurity-crs/*, asegurándose de trabajar con las reglas más recientes y actualizadas.

A continuación, se implementan estas reglas básicas en la configuración de Nginx. Para ello, se crea un nuevo directorio llamado modsec dentro de */etc/nginx/*, donde se alojará la configuración personalizada de ModSecurity. Dentro de este directorio, se genera un archivo de configuración llamado **modsecurity.conf**, basado en la configuración recomendada proporcionada por ModSecurity.

En este archivo, se establece correctamente el valor de *SecRuleEngine*, activando las reglas de detección y prevención. Esto permite que ModSecurity funcione en modo de protección activa, bloqueando solicitudes maliciosas en lugar de solo registrarlas.

```
GNU nano 7.2                                     /etc/nginx/modsec/modsecurity.conf *
# -- Rule engine initialization -----
# Enable ModSecurity, attaching it to every transaction. Use detection
# only to start with, because that minimises the chances of post-installation
# disruption.
#
SecRuleEngine On|
```

Figura 28: *modsecurity.conf*

Además de configurar ModSecurity, se crea un nuevo archivo de configuración llamado *main.conf* en el directorio */etc/nginx/modsec*. Este archivo actúa como el punto principal

donde se definen las reglas y configuraciones específicas que ModSecurity utilizará durante su operación.

El archivo *main.conf* contiene parámetros que aseguran que las reglas sean correctamente aplicadas y ajustadas según las necesidades del servidor.

```
GNU nano 7.2 main.conf *
Include /etc/nginx/modsec/modsecurity.conf
Include /usr/local/modsecurity-crs/crs-setup.conf
Include /usr/local/modsecurity-crs/rules/*.conf
```

Figura 29: *main.conf*

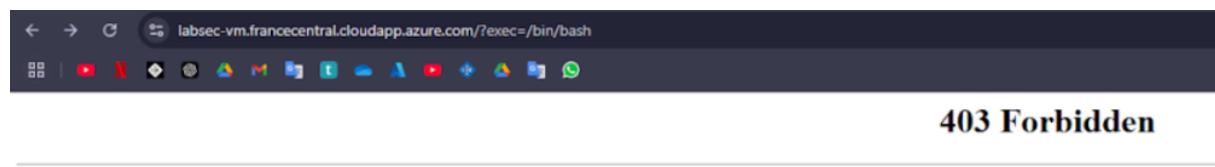
Los archivos clave para configurar ModSecurity y OWASP-CRS incluyen *modsecurity.conf* en */etc/nginx/modsec/*, que define la configuración base; *crs-setup.conf* en */usr/local/modsecurity-crs/*, donde se ajustan excepciones y sensibilidad del CRS; y el directorio *rules/* en la misma ubicación, que contiene las reglas específicas para distintos ataques y vulnerabilidades.

Para finalizar con este apartado, únicamente falta modificar el fichero de configuración */etc/nginx/sites-available/labsec-vm.francecentral.cloudapp.azure.com.conf*, añadiendo las siguientes líneas, como hemos explicado ya con anterioridad:

```
#ModSecurity
modsecurity on;
modsecurity_rules_file /etc/nginx/modsec/main.conf
```

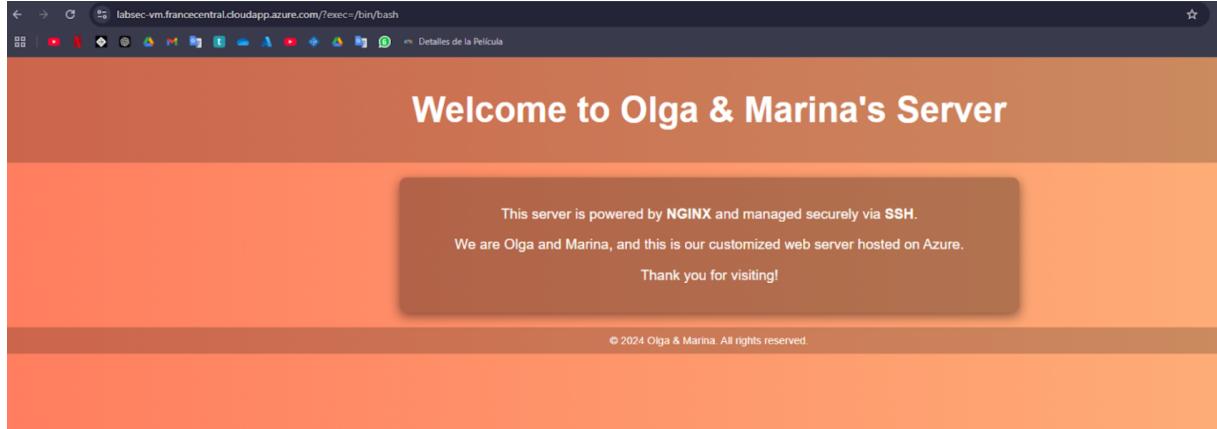
Figura 30: *labsec-vm.francecentral.cloudapp.azure.com.conf*

Para verificar que *ModSecurity* está funcionando correctamente, vamos a realizar la siguiente búsqueda en el navegador:



Así pues, podemos ver que se está activando una regla del OWASP-CRS para bloquear la solicitud. Al activarse dicha regla, *ModSecurity* intercepta la solicitud y devuelve un error **403 Forbidden**, que significa que la solicitud fue bloqueada por razones de seguridad. La razón por la cual la regla se ha activado es debido a que *ModSecurity* detecta que se está intentando usar el comando *exec*, que es típico en ataques de inyección de comandos.

Ahora bien, si desactivamos *ModSecurity*, Nginx procesa la solicitud como lo haría normalmente, es decir, sin analizar posibles patrones maliciosos. La solicitud se redirigirá al contenido del archivo *index.html* y se mostrará el contenido de la página web como se haría normalmente:



No configurar *ModSecurity* o desactivarlo podría ser peligroso porque permitiría que solicitudes maliciosas lleguen al servidor web sin ser bloqueadas.

Al no bloquear solicitudes con patrones como *exec=/bin/bash*, un atacante podría ejecutar comandos directamente en el sistema operativo del servidor. El atacante podría leer o modificar archivos críticos, escalar privilegios para tomar control total del servidor e instalar software malicioso.

3.2.8. Administración de tareas con Cron

En esta sección, pasaremos a explicar la herramienta *Cron*, un administrador de tareas que permite ejecutar comandos en un momento determinado. Para su configuración, hemos utilizado el archivo *crontab*, donde se listan todas las tareas que deben ejecutarse y el momento en el que deben hacerlo.

Hemos configurado tareas programadas que ejecutan comprobaciones diarias a las 7:00 AM con el objetivo de garantizar la seguridad y el correcto funcionamiento del sistema. Estas tareas incluyen:

- I. **Comprobación de actualizaciones diarias:** Se verifica y actualiza automáticamente los paquetes y dependencias del sistema para mantenerlo seguro y al día.
- II. **Monitorización del espacio en disco y uso de memoria:** Se registra el estado del disco y la memoria del sistema, asegurando un uso eficiente de los recursos.
- III. **Eliminación de registros de autenticación antiguos:** Se eliminan los archivos de registro de autenticación con más de 10 días de antigüedad para optimizar el almacenamiento y mantener la privacidad.
- IV. **Verificación de la ejecución de tareas:** Se comprueba que todas las tareas programadas se ejecuten correctamente, enviando un mensaje para validar su funcionamiento.

```
#VERIFICA NUEVAS ACTUALIZACIONES CADA DÍA A LAS 7:00 AM
0 7 * * * sudo apt update && sudo apt upgrade -y

#MUESTRA EL ESPACIO DISPONIBLE Y USO DE MEMORIA CADA DÍA A LAS 7:00 AM
0 7 * * * df -h >> /var/log/system_status.log && free -h >> /var/log/system_status.log

#LIMPIA LOS ARCHIVOS DE REGISTRO DE AUTENTICACIÓN CON MÁS DE 10 DÍAS CADA DÍA A LAS 7:00 AM
0 7 * * * find /var/log/auth.log* -type f -mtime +10 -exec rm -f {} \;

#COMPROBACIÓN DE QUE LAS TAREAS SE HAN EJECUTADO
0 7 * * * echo "Cron test ejecutado a $(date)" >> /var/log/cron_test.log
```

Figura 31: Tareas configuradas como usuario *root* mediante *Cron*

Para verificar que dichas reglas se ejecutan correctamente, podemos visualizar archivo de log *syslog*:

```
azureuser@vm:/var/log$ sudo tail -f /var/log/syslog
2024-12-18T16:34:01.082338+00:00 vm CRON[2700]: (root) CMD (df -h >> /var/log/system_status.log && free -h >> /var/log/system_status.log)
2024-12-18T16:34:01.082546+00:00 vm CRON[2701]: (root) CMD (sudo apt update && sudo apt upgrade -y)
2024-12-18T16:34:01.086606+00:00 vm CRON[2702]: (root) CMD (find /var/log/auth.log* -type f -mtime +10 -exec rm -f {} \;)
2024-12-18T16:34:01.086948+00:00 vm CRON[2703]: (root) CMD (echo "Cron test ejecutado a $(date)" >> /var/log/cron_test.log)
```

Figura 32: Ejecución de todas las tareas administradas por *Cron*

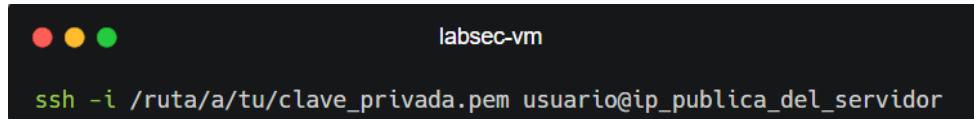
4. Puesta en producción

4.1. Manual para el acceso remoto al servicio

En este apartado explicaremos los pasos para acceder de forma remota con SSH a nuestro servicio, siendo necesario responder a los 3 factores de autenticación aplicados:

1) Conexión SSH y autenticación mediante clave pública.

Es necesario conectarse al servidor usando SSH con la siguiente línea de comando:



```
labsec-vm
ssh -i /ruta/a/tu/clave_privada.pem usuario@ip_publica_del_servidor
```

Donde:

- *-i /ruta/a/tu/clave_privada.pem*: Apunta a la ubicación de la clave privada local, que será compartida en la entrega del proyecto, junto a esta memoria, con el nombre **vm_key.pem**.
- *usuario*: Nombre de usuario utilizado para acceder, en este caso es **azureuser**.
- *ip_publica_del_servidor*: La dirección IP pública de la máquina virtual en Azure, siendo esta **20.199.93.120**.

2) Autenticación mediante la contraseña del usuario.

Una vez autenticado por clave pública, se pedirá el ingreso de la contraseña usada por el usuario utilizado, en este caso es **olgaymarina**.

3) Autenticación mediante un código de Google Authenticator.

Finalmente, para completar el proceso de autenticación, será necesario introducir el código de verificación generado por Google Authenticator.

Para que la aplicación comience a generar estos códigos, se debe hacer lo siguiente:

- 1) Acceder a la aplicación **Google Authenticator**.
- 2) Clickar en el menú situado en la parte superior izquierda.
- 3) Clickar en *Transferir códigos*.
- 4) Clickar en *Importar códigos*.
- 5) Clickar en *Scan QR Code* y escanear el siguiente código QR:



Figura 33: QR con la cuenta de Google Authenticator requerida

Esto es como se vería el acceso remoto al servicio ingresando los 3 factores de autenticación:

```
C:\Users\olgad>ssh -i C:\Users\olgad\Desktop\keys\vm_key.pem azureuser@20.199.93.120
(azureuser@20.199.93.120) Password:
(azureuser@20.199.93.120) Verification code:
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-1018-azure x86_64)
```

Figura 34: Acceso remoto al servicio

5. Conclusiones y futuras mejoras

5.1. Resumen de logros

Como conclusión haremos un pequeño resumen de los logros conseguidos:

- **Seguridad reforzada en SSH:** Implementación de autenticación en múltiples factores (clave pública, contraseña y TOTP) y protección contra ataques de fuerza bruta.
- **Seguridad del servidor web Nginx:** Uso de HTTPS, redirección segura, protección con Fail2ban, ModSecurity y bloqueo de acceso a archivos sensibles.
- **Control de tráfico y conexiones:** Configuración de límites de sesiones SSH y cierre de puertos innecesarios.
- **Automatización y mantenimiento:** Administración eficiente con tareas programadas usando Cron.

Gracias a estas configuraciones, se ha logrado un entorno seguro, optimizado y protegido frente a amenazas comunes en sistemas y servidores.

5.2. Propuestas de mejora

Para seguir reforzando la seguridad del servidor, se proponen las siguientes mejoras a implementar en el futuro:

- Cambio del puerto SSH predeterminado:

Actualmente, el servicio SSH escucha en el puerto 22, que es ampliamente conocido y frecuentemente atacado por bots y scripts automatizados. Una mejora sería configurar SSH para utilizar un puerto no estándar (por ejemplo, 2222 o 50022). Esto reducirá la exposición a ataques de fuerza bruta automatizados y exploraciones de puertos.

- Implementación de Port Knocking:

Se configuraría Port Knocking como una capa adicional de seguridad para proteger el acceso SSH. Este consiste en cerrar el puerto SSH por defecto y solo abrirlo temporalmente cuando se recibe una secuencia específica de conexiones a diferentes puertos. De esta manera, el puerto SSH permanecería invisible y solo podría ser accedido por usuarios que conozcan la secuencia correcta, lo que dificulta significativamente los intentos de intrusión.

6. Anexo

6.1. Referencias y enlaces consultados

- *DigitalOcean. How To Secure Nginx with Let's Encrypt on Ubuntu*

Guía utilizada para implementar un certificado SSL gratuito con Certbot y Let's Encrypt. Disponible en.

- *Linode. Securing Nginx with ModSecurity.*

Guía para configurar ModSecurity como firewall de aplicaciones web (WAF) en Nginx. Disponible en.

- *DigitalOcean. How To Set Up Multi-Factor Authentication for SSH on Ubuntu.*

Guía para configurar el factor de autenticación TOTP (Time-Based One-Time Password) usando Google Authenticator. Disponible en.

- *ChatGPT. Asistencia para configuración y redacción.*

Utilizado como herramienta para resolver preguntas técnicas concretas y como apoyo para la redacción de la documentación técnica. Disponible en.

- *Manual de Nginx.*

Guía para la correcta configuración del servidor web seguro. Disponible en.

- *Ejemplos realizados en clase.*

Se ha hecho uso de las configuraciones similares hechas en clase.