This module will describe how to secure a Linux operating system and workload in Azure

It will cover the following LAB topics:

1. Secure linux services using AppArmor profiles
2. Configure Linux firewall using IPtables and ufw
3. OS best practices for security
4. Securing SSH service
5. unattended upgrades for security patches
6. QoS (Quality of Service) for network traffic
7. Configure NGINX for HTTPS Grade A+ (SSL Labs)
8. Scan for vulnerabilities using Trivy

All of the commands below are meant to be run on a Ubuntu 24.04 LTS machine previously installed in the Azure-Infra Lab, using sudo or root privileges.

LAB 1: Secure linux services using AppArmor profiles

Intro: AppArmor is a Linux security module that provides Mandatory Access Control (MAC) for programs. It is similar to SELinux but uses a different approach to achieve the same goal of restricting the actions that a program can perform.

TASK: In this lab you are required to install and configure AppArmor to secure the operations of an NGINX web server.

Step 1: Check if AppArmor package is installed

Check if AppArmor is installed and check loaded AppArmor profiles using the command below:

```
sudo aa-status
```

Step 2: Check if AppArmor service is running

Using systemd, check if the AppArmor service is running using the command below:

```
sudo systemctl status apparmor
```

> **Warning:** If necessary, install apparmor and apparmor-utils packages.

```
sudo apt-get install -y apparmor apparmor-utils apparmor-profiles
```

Step 3: Create the AppArmor profile for NGINX

The configuration file for the NGINX AppArmor profile is located at `/etc/apparmor.d/usr.sbin.nginx`. You can create the profile using the command below:

```
sudo tee /etc/apparmor.d/usr.sbin.nginx > /dev/null <<EOF
#include <tunables/global>

/usr/sbin/nginx {
  #include <abstractions/base>
  #include <abstractions/apache2-common>

  capability net_bind_service,
  capability setgid,
  capability setuid,
  capability dac_override,
  capability dac_read_search,
  capability sys_chroot,

  /usr/sbin/nginx mr,
  /etc/nginx/** r,
  /usr/share/nginx/** r,
  /var/log/nginx/** rw,
  /var/lib/nginx/** rw,
  /run/nginx.pid w,

  /{,var/}run/nginx.pid w,
  /{,var/}run/nginx.lock w,

  /var/www/** r,
  /dev/urandom r,
  /dev/log w,
  /proc/** r,
  /sys/** r,

  deny /bin/dash rmix,
  deny /bin/bash rmix,
  deny /bin/sh rmix,
}
EOF
```

Step 4: Enable the NGINX AppArmor profile

Follow these steps to enable the NGINX AppArmor profile:

```
sudo apparmor_parser -r /etc/apparmor.d/usr.sbin.nginx
```

Step 5: Ensure the profile is in enforce mode

To ensure that the NGINX AppArmor profile is in enforce mode, follow the steps below:

```
sudo ln -s /etc/apparmor.d/usr.sbin.nginx /etc/apparmor.d/force-
complain/usr.sbin.nginx
sudo apparmor_parser -r /etc/apparmor.d/usr.sbin.nginx
```

Step 6: Install and enforce SSH AppArmor profile

```
wget https://sources.debian.org/data/main/a/apparmor/3.0.8-
3/profiles/apparmor/profiles/extras/usr.sbin.sshd \
    -O /etc/apparmor.d/usr.sbin.sshd

sudo apparmor_parser -av /etc/apparmor.d/usr.sbin.sshd
sudo aa-complain /etc/apparmor.d/usr.sbin.sshd
```

LAB 2: Configure Linux firewall using IPtables and ufw

Intro: Iptables is a user-space utility program that allows a system administrator to configure the IP packet filter rules of the Linux kernel firewall, implemented as different Netfilter modules. The filters are organized in different tables, which contain chains of rules for how to treat network traffic packets. Specifically on Ubuntu, UFW, or Uncomplicated Firewall, is a front-end for iptables and is particularly well-suited for host-based firewalls.

TASK: Install ufw and configure it to allow SSH, HTTP, and HTTPS connections, and to enable logging.

Step 1: Install ufw

In order to install ufw, run the following command:

```
sudo apt-get install ufw
```

Step 2: Allow SSH connections

To allow SSH connections, run the following command:

```
sudo ufw allow ssh comment "allow SSH connections"
```

Step 3: Allow HTTP and HTTPS connections

For HTTP and HTTPS connections, run the following commands:

```
sudo ufw allow http comment "allow HTTP connections"
sudo ufw allow https comment "allow HTTPS connections"
```

Check the status of ufw rules:

```
sudo ufw status
```

Step 4: Enable logging

To enable logging, run the following command:

```
sudo ufw logging on
```

Step 5: Enable ufw

Finally, enable ufw at boot and start the firewall using the following command:

```
sudo ufw enable
```

LAB 3: OS best practices for security

Intro: Securing a Linux operating system involves implementing best practices to protect the system from unauthorized access, data breaches, and other security threats. This includes configuring user accounts, setting up firewalls, enabling automatic updates, and implementing other security measures to safeguard the system and its data.

TASK: Implement OS best practices for security, including setting up automatic unattended updates, configuring password complexity and expiration, setting up proper time synchronization, and configure tools for better system entropy, set sudo password, and make sure everything listens only on localhost.

Step 1: update and setup automatic unattended updates

Enable ubuntu unattended updates, this configuration file provides various settings for the Unattended-Upgrade utility, allowing you to customize how the system handles automatic package upgrades, rebooting, email notifications, and cleanup of unused packages and dependencies.

```
sudo cat <<EOF > /etc/apt/apt.conf.d/50unattended-upgrades
Unattended-Upgrade::Allowed-Origins {
        "${distro_id}:${distro_codename}";
        "${distro_id}:${distro_codename}-security";
        "${distro_id}ESMApps:${distro_codename}-apps-security";
        "${distro_id}ESM:${distro_codename}-infra-security";
        "${distro_id}:${distro_codename}-updates";
};
Unattended-Upgrade::Package-Blacklist {
```

```
};
Unattended-Upgrade::DevRelease "false";
Unattended-Upgrade::AutoFixInterruptedDpkg "true";
Unattended-Upgrade::MinimalSteps "true";
Unattended-Upgrade::Remove-Unused-Kernel-Packages "true";
Unattended-Upgrade::Automatic-Reboot "false";
Unattended-Upgrade::Automatic-Reboot-WithUsers "false";
Unattended-Upgrade::Automatic-Reboot-Time "02:00";
Unattended-Upgrade::Mail "some-email@example.com";
Unattended-Upgrade::MailReport "always";
Unattended-Upgrade::Remove-New-Unused-Dependencies "true";
Unattended-Upgrade::Remove-Unused-Dependencies "true";
EOF
```

The lines of text that follow are configuration settings for the APT (Advanced Package Tool) package manager. These settings control the automatic updates and upgrades of packages on a system. The configuration file /etc/apt/apt.conf.d/20auto-upgrades is used to specify the settings for automatic updates and upgrades. The settings include the following:

```
sudo cat <<EOF > /etc/apt/apt.conf.d/20auto-upgrades
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Unattended-Upgrade "1";
APT::Periodic::Download-Upgradeable-Packages "1";
APT::Periodic::AutocleanInterval "3";
EOF
```

The bash command systemctl restart unattended-upgrades.service is used to restart the unattended-upgrades.service service.

```
sudo systemctl restart unattended-upgrades.service
```

Step 2: Configure password complexity

```
sudo apt-get install libpam-pwquality
sudo sed -i 's/# minlen = 8/minlen = 12/' /etc/security/pwquality.conf
```

Step 3: Password expiration

In linux systems, password expiration policies are used to enforce regular password changes for user accounts. This helps to enhance security by reducing the risk of unauthorized access due to compromised or weak passwords. By setting password expiration policies, users are required to change their passwords at regular intervals, which can help prevent unauthorized access and improve overall system security.

```

```

```
sudo sed -i 's/PASS_MAX_DAYS\t99999/PASS_MAX_DAYS\t90/' /etc/login.defs
sudo sed -i 's/PASS_MIN_DAYS\t0/PASS_MIN_DAYS\t7/' /etc/login.defs
sudo sed -i 's/PASS_WARN_AGE\t7/PASS_WARN_AGE\t14/' /etc/login.defs
```

Step 4: Setup proper time synchronization

Chrony is a time synchronization daemon in Linux systems that helps maintain accurate and synchronized time across the network. It is important to have good timekeeping in a system as it ensures proper functioning of various critical processes, such as authentication, logging, and distributed systems coordination. Chrony uses reliable time sources, such as NTP servers, to obtain accurate time information and adjusts the system clock accordingly, minimizing time discrepancies and ensuring consistent time across the network.

```
sudo echo bindaddress 127.0.0.1 >> /etc/chrony/chrony.conf
sudo echo bindaddress ::1 >> /etc/chrony/chrony.conf
```

```
sudo systemctl restart chrony
```

Step 5: Install and haveged the random number generator

haveged is a software solution that provides a random number generator for Linux systems. It is important for a system to have enough entropy available because random numbers are crucial for various security-related operations, such as generating cryptographic keys and ensuring secure communication. Insufficient entropy can lead to weak or predictable random numbers, which can compromise the security of the system. haveged helps to increase the available entropy by collecting environmental noise and using it to generate random numbers. This ensures that the system has a sufficient amount of randomness for secure operations.

Check current system entropy:

```
cat /proc/sys/kernel/random/entropy_avail
```

Step 6: Configure pollinate

Pollinate is a service in Ubuntu systems that helps to increase the available entropy for generating random numbers. Entropy is crucial for various security-related operations, such as generating cryptographic keys and ensuring secure communication. Insufficient entropy can lead to weak or predictable random numbers, compromising the security of the system. Pollinate helps to increase the available entropy by collecting random data from external sources and using it to generate random numbers. This ensures that the system has a sufficient amount of randomness for secure operations.

Remove the existing Pollinate cache:

```
sudo rm -rf /var/cache/pollinate
sudo systemctl restart pollinate
```

Step 7: configure haveged and rng-tools5

Rng-tools is a set of utilities for managing hardware random number generators. It provides a daemon that collects entropy from various sources and makes it available to the kernel as a random number generator.

```
sudo apt-get install -y haveged rng-tools5
sudo systemctl enable --now haveged
sudo systemctl enable --now rng-tools5
```

Step 8: Configure haveged number generator

The default settings for haveged are not enough for a production system. We need to increase the number of bits collected per interrupt and the number of interrupts per second.

```
sudo sed -i 's/DAEMON_ARGS="-w 1024"/DAEMON_ARGS="-w 2048 -n 0 -v 1"/'
/etc/default/haveged
```

Step 9: Tests for entropy

In order to test the entropy of the system, you can use the following commands:

```
dd if=/dev/random of=random_output count=8192

rngtest < random_output
```

Step 10: Enable sudo password

Enabling password for sudo is a good practice to ensure that only authorized users can execute privileged commands. To enable password for sudo, you can modify the sudo configuration file (`/etc/sudoers`) with the following settings:

```
sudo apt update
sudo apt install pwgen -y
```

Let's generate a random passwords and set it for our admin user.

```
NEW_PW=$(pwgen 64 1) ; echo $NEW_PW ; echo "azureadmin:${NEW_PW}" | chpasswd
```

The recommended way to change sudo settings is to use visudo but changing the files directly is acceptable too as long as we check the syntax after. So, the users created by cloud-init VM_USERNAME='azureadmin' is both part of the sudo group and has the NOPASSWD attribute for all commands set in the file /etc/sudoers.d/90-cloud-init-users. The command below has a risk attached as it does a direct no-backup, in-place delete of NOPASSWD and will rely on the fact that user has the password set previously.

```
sudo sed -i 's/NOPASSWD\://g' /etc/sudoers.d/90-cloud-init-users

visudo -cf /etc/sudoers.d/90-cloud-init-users
```

To check the new permissions for user azureadmin.

```
sudo -l -U azureadmin
```

Step 11: Make sure everything listens only on localhost

Unless otherwise stated no other services should be listening on public reachable interface. This is a good practice to ensure that only the necessary services are exposed to the public network and that the attack surface is minimized.

```
sudo ss -tulnp
```

```
Netid             State              Recv-Q             Send-Q
Local Address:Port                          Peer Address:Port
Process
udp               UNCONN             0                  0
127.0.0.1:53                                 0.0.0.0:*                 users:
(("dnsmasq",pid=106,fd=4))
udp               UNCONN             0                  0
127.0.0.1:323                                0.0.0.0:*
udp               UNCONN             0                  0
[::1]:323                                   [::]:*
tcp               LISTEN             0                  511
127.0.0.1:41149                              0.0.0.0:*                 users:
(("node",pid=2616,fd=19))
tcp               LISTEN             0                  32
127.0.0.1:53                                 0.0.0.0:*                 users:
(("dnsmasq",pid=106,fd=5))
```

LAB 4: Securing SSH service

Intro: Open-SSH is a widely used tool for remote access to Linux systems. It is important to secure the SSH service to prevent unauthorized access and protect sensitive data. This includes using strong encryption,

disabling password authentication, and limiting access to specific users and groups.

TASK: Secure the SSH service by configuring it to listen on a specific interface, restrict to IPv4, limit the number of sessions, disable key forwarding, reduce the timeout, and allow only certain groups.

Step 1: Configure the SSH server

To configure the SSH server to listen on a specific interface, restrict to IPv4, limit the number of sessions, disable key forwarding, reduce the timeout, and allow only certain groups, you can modify the SSH configuration file (`/etc/ssh/sshd_config`) with the following settings:

```
sudo cat <<EOF > /etc/ssh/sshd_config
#
# Set protocol
#
Protocol 2

#
# Set interface and listening ports
#
ListenAddress 0.0.0.0:22
HostKey /etc/ssh/ssh_host_ecdsa_key
HostKey /etc/ssh/ssh_host_ed25519_key

#
# Log levels
#
LogLevel VERBOSE
SyslogFacility AUTH

#
# Do not allow Root login
#
PermitRootLogin no

#
# Use Pubkey auth
#
PubkeyAuthentication yes
AuthenticationMethods publickey

#
# List of accepted algorithms, chipers, MACs
#
HostKeyAlgorithms ssh-ed25519,ssh-rsa
PubkeyAcceptedKeyTypes sk-ecdsa-sha2-nistp256@openssh.com,sk-ssh-
ed25519@openssh.com,ssh-rsa
KexAlgorithms curve25519-sha256@libssh.org,ecdh-sha2-nistp521,ecdh-sha2-
nistp384,ecdh-sha2-nistp256,diffie-hellman-group-exchange-sha256
Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-
gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr
MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com,umac-128-
```

```
etm@openssh.com,hmac-sha2-512,hmac-sha2-256,umac-128@openssh.com

#
# Deny all other auth methods
#
UsePAM no
PasswordAuthentication no
KbdInteractiveAuthentication no
StrictModes yes
IgnoreRhosts yes
IgnoreUserKnownHosts yes
HostbasedAuthentication no
PermitEmptyPasswords no
KerberosAuthentication no
KerberosOrLocalPasswd no
KerberosTicketCleanup yes
GSSAPIAuthentication no
GSSAPICleanupCredentials no
ChallengeResponseAuthentication no

#
# Login session setup
#
LoginGraceTime 10s
PerSourceMaxStartups 1
MaxAuthTries 1
MaxStartups 10:30:60
MaxSessions 5
ClientAliveInterval 300
ClientAliveCountMax 0

#
# Deny all kind forwarding or tunnels
#
PermitUserEnvironment no
AllowAgentForwarding no
AllowTcpForwarding no
PrintMotd no
TCPKeepAlive no
AcceptEnv LANG LC_*
PermitTunnel no
GatewayPorts no
X11Forwarding no
X11UseLocalhost no
PrintLastLog no
DebianBanner no
Compression no
DisableForwarding yes
PermitListen none
PermitOpen none
PermitTunnel no

#
# subsystem for sftp
```

```
#
Subsystem sftp /usr/lib/openssh/sftp-server -f AUTHPRIV -l INFO

#
# groups that are not allowed to ssh
#
DenyGroups deny-ssh
EOF
```

After making these changes, save the file and restart the SSH service for the changes to take effect:

```
sudo systemctl restart sshd
```

Step 2: Generate stronger servers keys

```
sudo ssh-keygen -t rsa -b 4096 -f /etc/ssh/ssh_host_rsa_key
sudo ssh-keygen -t ed25519 -f /etc/ssh/ssh_host_ed25519_key
```

LAB 5: QoS (Quality of Service) for network traffic

Intro: Quality of Service (QoS) is a set of technologies and techniques used to manage network traffic and ensure that critical applications receive the necessary bandwidth and resources to function properly. QoS can help improve network performance, reduce latency, and ensure a consistent user experience for applications and services.

TASK: Configure QoS settings in Linux to optimize network traffic for a virtual machine running in Azure.

Step 1: Check the current QoS settings

First of all check the current QoS settings in linux for your VM this consist of IPv4 Kernel buffer settings and congestion control algorithm.

```
sysctl -a | egrep "
(net.core.rmem_max|net.core.wmem_max|net.ipv4.tcp_rmem|net.ipv4.tcp_wmem|net.ipv4.
tcp_congestion_control)"
```

```
net.core.rmem_max = 212992
net.core.wmem_max = 212992
net.ipv4.tcp_rmem = 4096        131072  6291456
net.ipv4.tcp_wmem = 4096        16384   4194304
net.ipv4.tcp_congestion_control = cubic
```

Step 2: Configure QoS settings

Recommended settings including congestion control algorithm BBR:

```
sudo cat <<EOF > /etc/sysctl.d/99-azure-network-buffers.conf

net.core.rmem_max = 2147483647
net.core.wmem_max = 2147483647
net.ipv4.tcp_rmem = 4096 67108864 1073741824
net.ipv4.tcp_wmem = 4096 67108864 1073741824
net.ipv4.tcp_congestion_control = bbr
EOF

sudo sysctl -p /etc/sysctl.d/99-azure-network-buffers.conf
```

To view the qdisc settings on linux use the following command:

```
tc qdisc show
```

```
qdisc mq 0: dev eth0 root
qdisc fq_codel 0: dev eth0 parent :2 limit 10240p flows 1024 quantum 1514 target
5ms interval 100ms memory_limit 32Mb ecn drop_batch 64
qdisc fq_codel 0: dev eth0 parent :1 limit 10240p flows 1024 quantum 1514 target
5ms interval 100ms memory_limit 32Mb ecn drop_batch 64
```

LAB 6: Configure NGINX for HTTPS Grade A+ (SSL Labs)

Intro: Securing a website with HTTPS is essential to protect sensitive data and ensure the privacy and security of users. HTTPS encrypts the data transmitted between the web server and the client, preventing

TASK: Configure NGINX to use HTTPS with a strong SSL configuration that achieves an A+ grade on SSL Labs, using a certificate generated with certbot, and a Diffie-Hellman key.

Step 1: Generate a strong SSL configuration using snapd and certbot

```
sudo snap install core; sudo snap refresh core
sudo snap install --classic certbot
sudo ln -s /snap/bin/certbot /usr/bin/certbot
```

Now let's create a certificate for your Azure Public IP address based on your Azure Domain Name Label.

```
certbot --nginx -d <domainnamelabel>.<location>.cloudapp.azure.com # replace with
your domain name
```

Step 2: Create a strong SSL configuration for NGINX

Start with a template using the Mozilla SSL Configuration Generator, for example this is a link to a intermediate configuration for NGINX. The highlights of this config file is using lates version of TLS, HSTS, OCSP stapling, and a modern cipher suite as well as HTTP2.

```
cat <<EOF >
# generated 2024-08-26, Mozilla Guideline v5.7, nginx 1.17.7, OpenSSL 1.1.1k,
intermediate configuration
# https://ssl-
config.mozilla.org/#server=nginx&version=1.17.7&config=intermediate&openssl=1.1.1k
&guideline=5.7
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;

    ssl_certificate /path/to/signed_cert_plus_intermediates; #
/etc/letsencrypt/live/<domainnamelabel>.
<location>.cloudapp.azure.com>/fullchain.pem;
    ssl_certificate_key /path/to/private_key; #
/etc/letsencrypt/live/<domainnamelabel>.
<location>.cloudapp.azure.com>/privkey.pem;
    ssl_session_timeout 1d;
    ssl_session_cache shared:MozSSL:10m;  # about 40000 sessions
    ssl_session_tickets off;

    # curl https://ssl-config.mozilla.org/ffdhe2048.txt > /path/to/dhparam
    ssl_dhparam /etc/ssl/dhparam.pem;
    ssl_ecdh_curve secp384r1;

    # intermediate configuration
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-
ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-
POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-
SHA384:DHE-RSA-CHACHA20-POLY1305;
    ssl_prefer_server_ciphers off;

    # HSTS (ngx_http_headers_module is required) (63072000 seconds)
    add_header Strict-Transport-Security "max-age=63072000" always;

    # OCSP stapling
    ssl_stapling on;
```

```
    ssl_stapling_verify on;

    # replace with the IP address of your resolver
    resolver 127.0.0.53;
}
EOF
```

Step 4: Create a stronger Diffie-Hellman

To create a Diffie-Hellman using the algorithm secp384r1 key use the following commands.

```
openssl dhparam -out /etc/ssl/dhparam.pem 4096
```

Then add the following lines to the /etc/nginx/nginx.conf file's http {} block:

```
ssl_dhparam /etc/ssl/dhparam.pem; # Use our custom strong Diffie-Hellman for
handshakes
ssl_ecdh_curve secp384r1; # Secure it with an elliptic curve algorithm instead of
RSA
```

We now need to restart the NGINX service to apply the changes.

```
sudo systemctl restart nginx
```

Step 5: Testing the SSL configuration using Qualys SSL Labs

The first option is to navigate to the website Qualys SSL Labs and enter the domain name of your website. The website will then run a test on your website's SSL configuration and provide you with a grade. Qualys SSL Labs provides a comand line tool as well that can be found here SSL Labs API.

LAB 7: Scan for vulnerabilities using Trivy

Intro: Triy is a simple and comprehensive vulnerability scanner for containers and other artifacts. A software vulnerability is a security flaw or weakness in a software application that can be exploited by an attacker to gain unauthorized access to a system or data. Vulnerabilities can exist in various components of a software application, including the code, libraries, configurations, and dependencies.

TASK: Scan the running Ubuntu host for vulnerabilities using Trivy.

Step 1: Install Trivy

To install Trivy, you can use the following commands:

```

```

```
sudo apt-get install wget apt-transport-https gnupg
wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | gpg --
dearmor | sudo tee /usr/share/keyrings/trivy.gpg > /dev/null
echo "deb [signed-by=/usr/share/keyrings/trivy.gpg]
https://aquasecurity.github.io/trivy-repo/deb generic main" | sudo tee -a
/etc/apt/sources.list.d/trivy.list
sudo apt-get update
sudo apt-get install trivy
```

Step 2: Scan the Ubuntu host for vulnerabilities

To scan the Ubuntu host for vulnerabilities using Trivy, you can use the following command:

```
sudo trivy rootfs --scanners vuln --severity HIGH,CRITICAL /
```

This command will scan the root filesystem of the Ubuntu host for vulnerabilities with a severity level of HIGH or CRITICAL.