Task 1

Containers:

Containers provide isolated environments within a shared OS kernel, primarily using namespaces and cgroups to separate applications. They are lightweight and enable efficient resource usage, making them ideal for scalable applications. However, they share the host's OS kernel, which can be a security vulnerability if an attacker breaks out of the container. Security capabilities include image signing and verification, runtime security policies, and access controls to minimize unauthorized access. Nevertheless, containers are limited in their isolation compared to virtual machines, as they do not have a separate OS kernel, increasing risks associated with kernel vulnerabilities.

Virtualization:

Virtualization technology, such as hypervisors, allows multiple virtual machines (VMs) to run on a single physical host with strong isolation. Each VM operates with its OS, providing a secure boundary between them and minimizing the risk of cross-VM attacks. Virtualization's security strengths lie in this isolation and in VM escape prevention mechanisms, making it a strong choice for applications requiring high-security levels. However, hypervisors can be a single point of failure, and attacks targeting the hypervisor could impact all VMs on the host. Additionally, VMs tend to have higher resource overhead than containers, potentially reducing scalability.

Task 2

Introduction

In the current landscape of supply chain threats, securing the production and distribution channels of networking equipment is crucial. The recent rise in sophisticated supply chain attacks has highlighted the importance of not only in-house security practices but also rigorous control over third-party actors involved in the supply chain. This report outlines actionable steps to secure the supply chain for a networking hardware and software company, including specific measures for four critical actors: part suppliers, in-house and outsourced employees, transportation companies, and firmware maintenance providers.

Actors and Points of Concern

1. ***Part Suppliers Concerns:***

   Parts used in networking equipment, such as microchips or circuit boards, are often outsourced and manufactured internationally. These components could be tampered with during manufacturing, enabling malicious actors to introduce backdoors or vulnerabilities at the hardware level.

Actionable Measures:

Trusted Supplier Certification: Establish a trusted list of suppliers with stringent security certifications (e.g., ISO/IEC 27001) and enforce regular audits to ensure adherence to security standards.

TPM Integration: Require suppliers to incorporate Trusted Platform Module (TPM) chips in critical components, allowing for hardware-based cryptographic verification that can detect unauthorized alterations.

Potential Challenges:

Establishing and enforcing supplier certifications may increase costs and reduce supplier options, which could impact production timelines.

2. In-House and Outsourced Employees Concerns:

Employees, especially those involved in the software and firmware development process, could be targeted for insider attacks or manipulated into introducing malicious code or vulnerabilities.

Actionable Measures:

EDR and UBA Tools: Implement Endpoint Detection and Response (EDR) and User Behavioral Analytics (UBA) to monitor abnormal activities and quickly identify potential insider threats or compromised accounts.

Access Control and Code Reviews: Restrict access to sensitive components and conduct regular peer-reviewed code audits to identify malicious or anomalous code changes.

Potential Challenges:

Monitoring employee behavior requires balancing privacy concerns, and continuous code review could slow down development processes, potentially impacting product release timelines.

3. Transportation Companies Concerns:

Once products leave the manufacturing facility, there is a risk of tampering during transit, especially if products are shipped internationally. Malicious actors could intercept shipments and implant spyware or other malicious elements.

Actionable Measures:

Tamper-Evident Packaging: Use tamper-evident seals and digitally track packaging conditions to detect any unauthorized access during transit.

NDR at Facilities: Employ Network Detection and Response (NDR) systems at logistics hubs and storage facilities to detect any unauthorized access attempts to network-connected devices.

Potential Challenges: Implementing tamper-evident solutions and ensuring proper digital tracking may increase logistical costs and require training for logistics personnel.

4. Firmware Maintenance Providers Concerns:

Firmware updates are critical for network devices, but they also present a significant risk. Malicious firmware updates can introduce backdoors or other vulnerabilities that compromise the device.

Actionable Measures:

Code Signing and Verification: Mandate that all firmware updates are digitally signed, with end devices verifying signatures before accepting any updates.

Secure Update Distribution Channels: Ensure that updates are distributed only through secure, verified channels, preventing unauthorized access to the firmware distribution process.

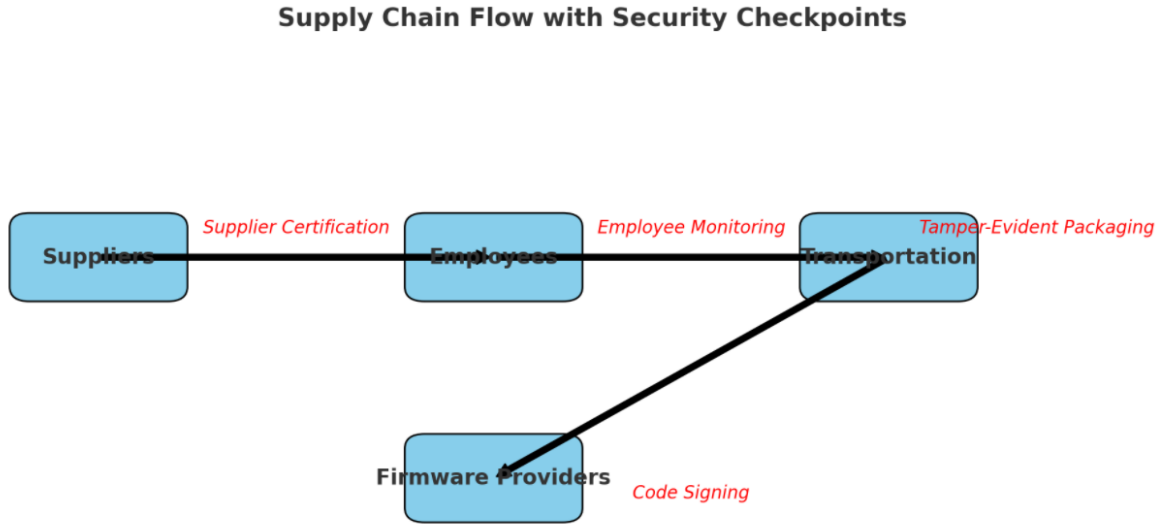
Potential Challenges: The requirement for secure code signing and distribution channels may limit flexibility, particularly when working with multiple vendors or third-party developers.

Additional Considerations

1. Regular Penetration Testing and Audits: Conducting regular penetration testing on devices and audits on the entire supply chain can help detect vulnerabilities before they are exploited by attackers. For example, simulating potential attacks on transportation facilities or suppliers can provide valuable insights into security gaps.
2. Threat Intelligence Sharing: Establish communication channels with suppliers and vendors to share threat intelligence about the latest supply chain attack vectors. This can foster collaboration and improve security postures across the supply chain.
3. Automated Security Alerts and Response: Use automation to monitor the supply chain continuously and trigger alerts when anomalies are detected. Automated response systems can prevent potential breaches before they escalate.

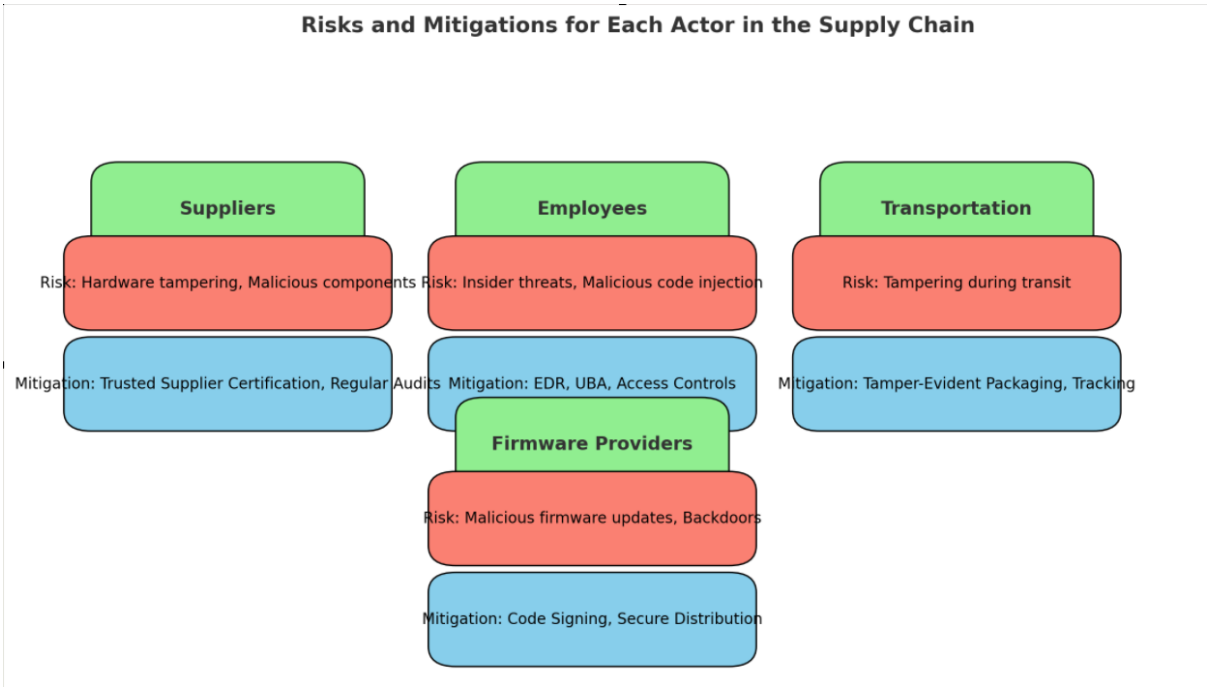
Visual Representations

1. Supply Chain Flowchart:



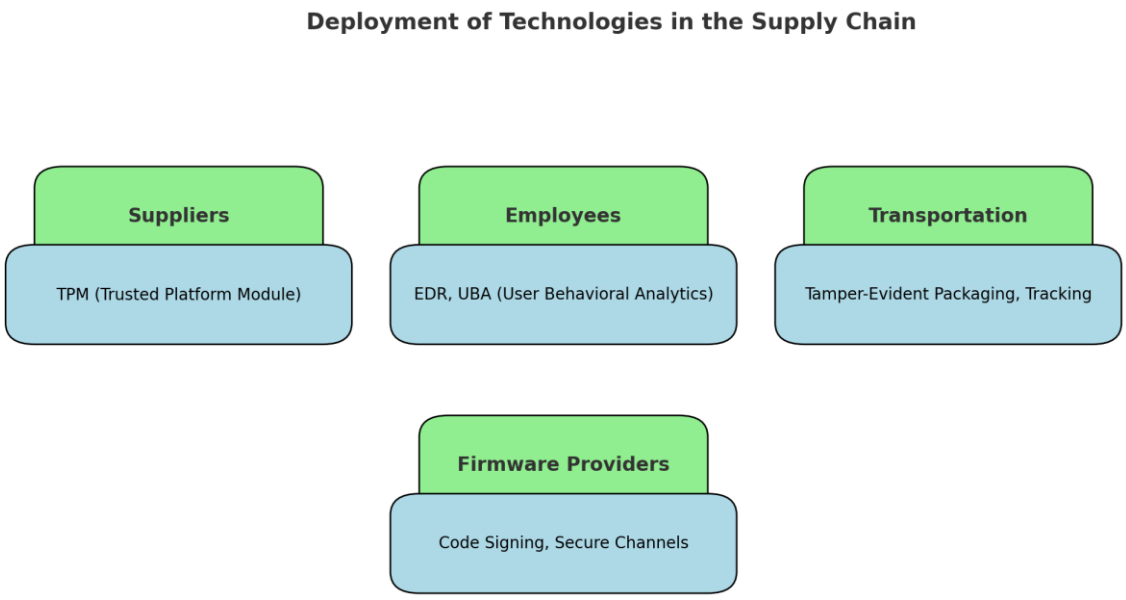
This diagram visually represents the flow of components and actions, with each security checkpoint labeled in red.

2. Risk Analysis Matrix:



This diagram highlights the risks and mitigations in a structured manner, allowing for clear identification of potential security measures needed for each actor in the supply chain.

3. *Security Technology Deployment Diagram*



This diagram illustrates where different security technologies are deployed in the supply chain to ensure comprehensive coverage. Each technology is strategically applied to address specific security concerns for each actor in the supply chain.

Task 3A

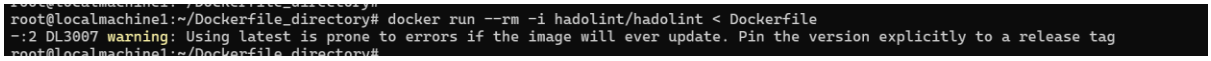
Dockerfile Linting Report

Tool Used

- Linter: Hadolint
- Command: docker run --rm -i hadolint/hadolint < Dockerfile

Initial Issue Detected

1. ***DL3007: Warning - Using latest is prone to errors if the image ever updates. It is recommended to pin the version explicitly to a release tag.***

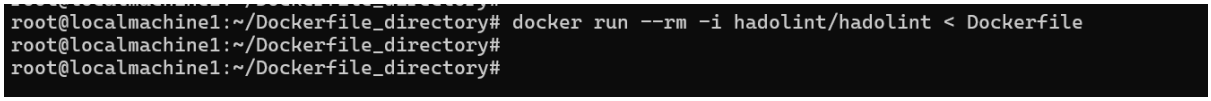


Fix Applied

- Replaced FROM python:latest with FROM python:3.9-slim in the Dockerfile to ensure consistency and prevent unexpected changes from latest updates.

Final Linting Output

- No issues found.



Files Attached (if required)

1. ***Dockerfile Before Fixes (if you saved a copy of the original)***
2. ***Dockerfile After Fixes***
3. ***Hadolint Output:***
    - ***Before Fix:***

Use the official Python image from Docker Hub

FROM python:latest

Set the working directory in the container

WORKDIR /app

Copy the requirements file into the container

COPY requirements.txt .

Install any dependencies specified in requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

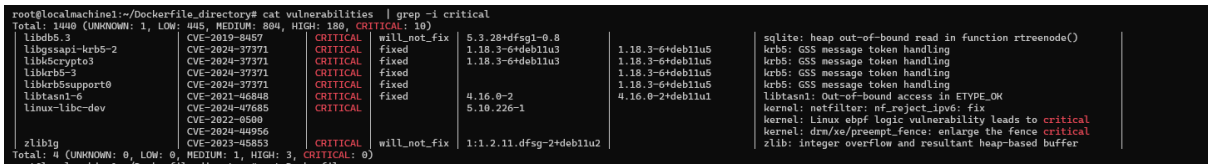
```
# Copy the current directory contents into the container at /app
COPY . .

# Expose port 5000 for the Flask app
EXPOSE 5000

# Define the command to run the application
CMD ["python", "app.py"]
```

- ***After Fix:***

```
# Use the official Python image from Docker Hub
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file into the container
COPY requirements.txt .

# Install any dependencies specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy the current directory contents into the container at /app
COPY . .

# Expose port 5000 for the Flask app
EXPOSE 5000

# Define the command to run the application
CMD ["python", "app.py"]
```

TASK3B

Vulnerability report



Summarizing the Vulnerabilities:

- Total Vulnerabilities: Summarize the count of vulnerabilities by severity (Total: 1440 (UNKNOWN: 1, LOW: 445, MEDIUM: 804, HIGH: 180, CRITICAL: 10).
- Total Vulnerabilities in Python Packages: Mention any critical/high vulnerabilities in key libraries like Flask, pip, and setuptools, as shown in the output.

Identifying Key Vulnerabilities to Address:

- Highlight any CRITICAL or HIGH vulnerabilities that could impact security, such as:
    - zlib1g with CVE-2023-45853 (CRITICAL): Potential buffer overflow.
    - Flask with CVE-2023-30861 (HIGH): Possible disclosure of session cookies.

Explanation of Changes

Upgrade Debian Base Packages: apt-get upgrade -y will apply any available security patches to the Debian base image, which should address vulnerabilities in libkrb5, libtasn1-6, zlib1g, and related libraries.

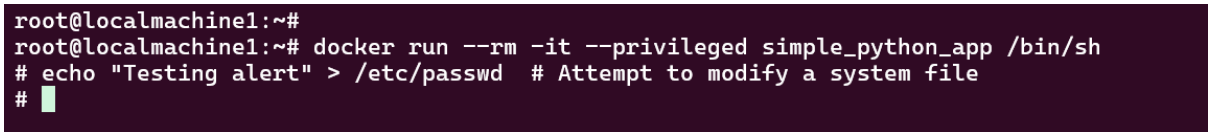
Explicitly Install Specific Libraries: Installing libkrb5-3, libkrb5support0, libk5crypto3, libgssapi-krb5-2, and zlib1g ensures that we pull the latest patched versions directly.

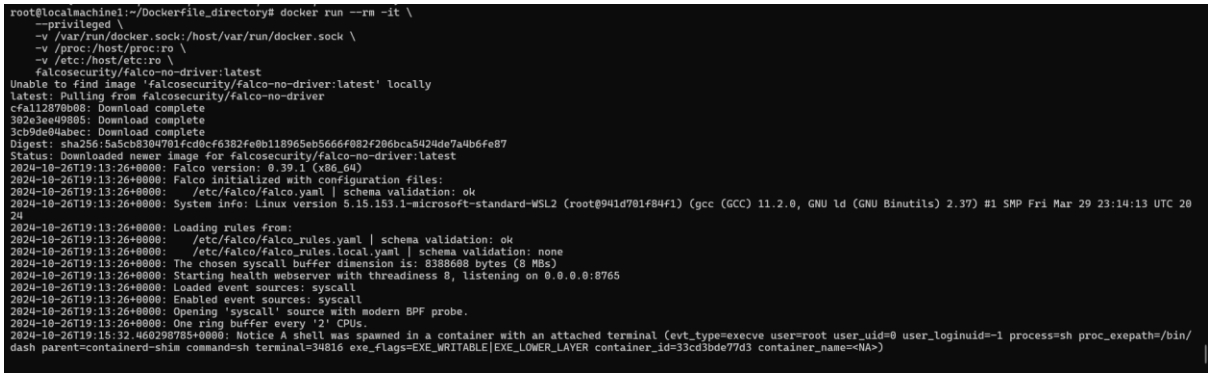
Remove Build Dependencies: The gcc compiler is temporarily installed to compile any dependencies that might need it and is then removed to keep the image lean and reduce vulnerabilities.

Clear Apt and Pip Caches: This step reduces the image size and removes unnecessary files that could have potential vulnerabilities.

After the changes

TASK3C





From screenshot, it appears that Falco successfully detected the activity and generated an alert.

runtime security scanner you used:

Falco.

The image used:

falcosecurity/falco-no-driver

(Pulled from Docker Hub).

Commands and activities used to trigger the alerts:

Falco Run Command:

bash

```
docker run --rm -it \
  --privileged \
  -v /var/run/docker.sock:/host/var/run/docker.sock \
  -v /proc:/host/proc:ro \
  -v /etc:/host/etc:ro \
  falcosecurity/falco-no-driver:latest
```

Trigger Command inside the Container:

```
echo "Testing alert" > /etc/passwd  # Attempt to modify a system file
```

This command attempts to write to /etc/passwd, a suspicious action, triggering Falco's alert as seen in the screenshot.

Additional Observations:

Falco detects when a shell is opened in a privileged container and logs the activity, helping in identifying potential threats.