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 highsecurity 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 peerreviewed 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.

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.

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

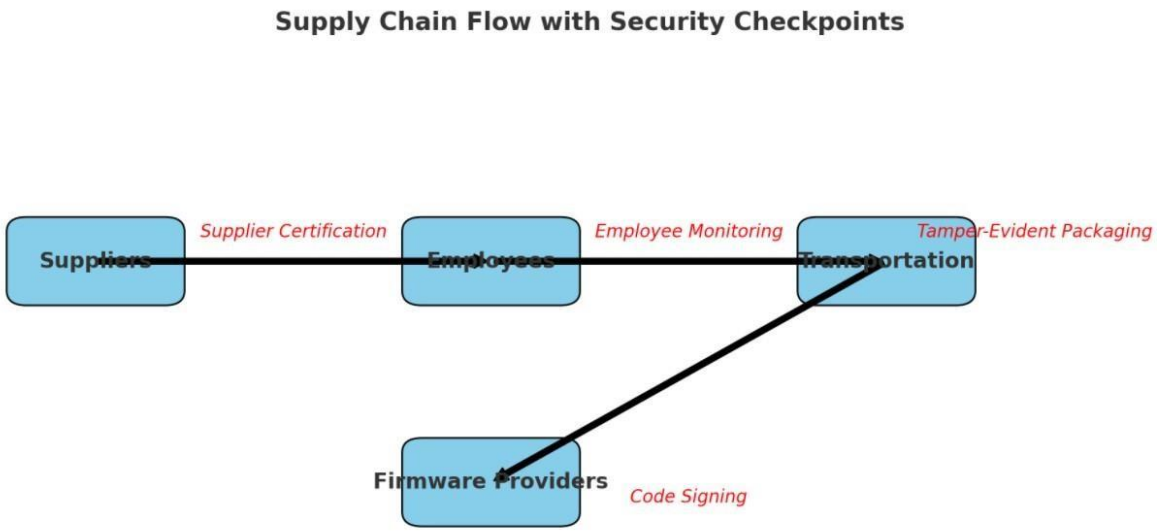
Regular Penetration Testing and Audits: Conducting regular penetration testing on devices and audits on the entire supply chain can help detect vulnerabilities before they are exploited by attackers. For example, simulating potential attacks on transportation facilities or suppliers can provide valuable insights into security gaps.

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.

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.

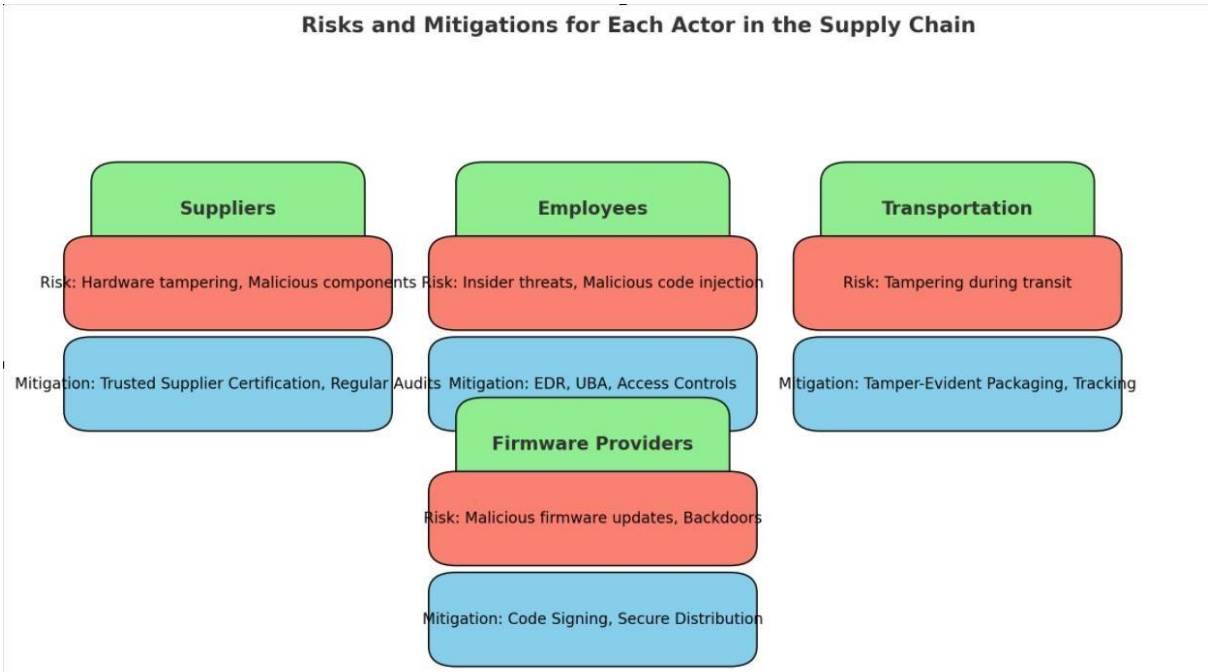
Visual Representations

Supply Chain Flowchart:



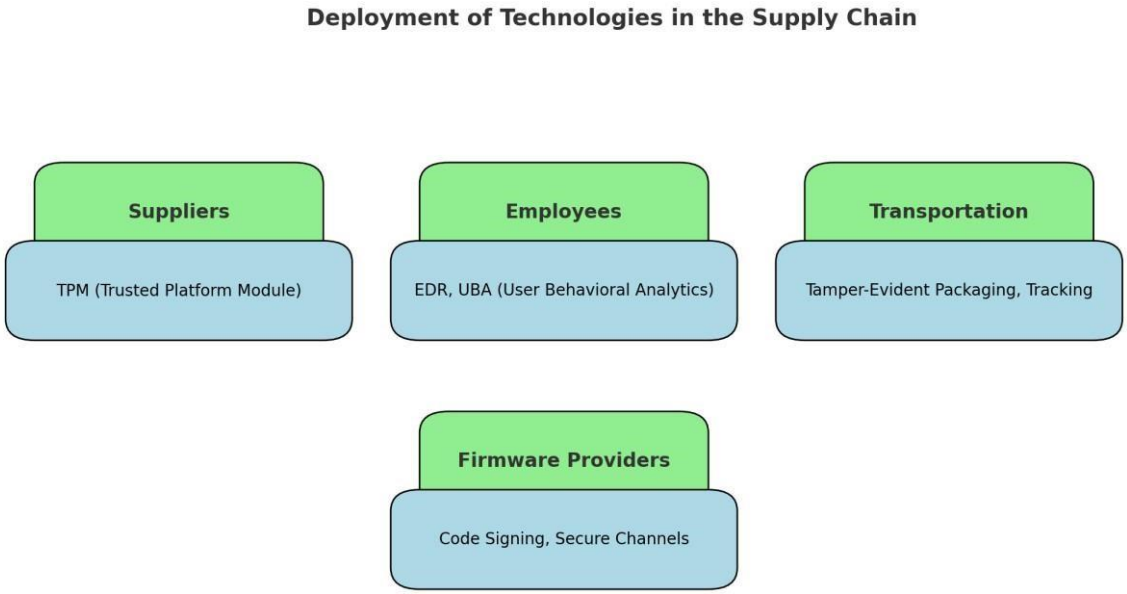
This diagram visually represents the flow of components and actions, with each security checkpoint labeled in red.

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.

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

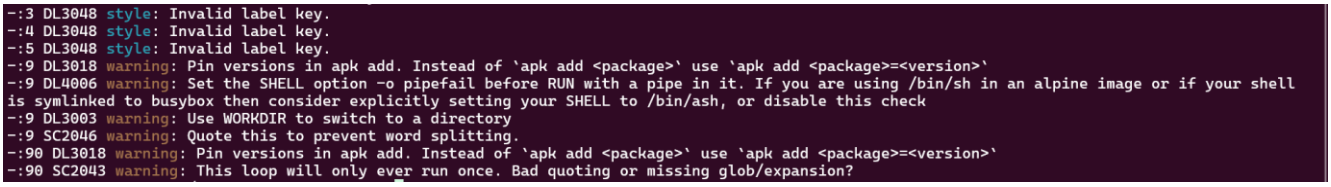
Dockerfile Linting Report

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

Set Working Directory (DL3003):

Added WORKDIR /usr/src/app to specify the working directory explicitly. This is a best practice as it makes file handling consistent and eliminates the need to cd into directories manually.

Pin Package Versions (DL3018):

Specified versions for apk packages (libstdc++=12.2.1-r0, curl=7.88.0-r0, gnupg=2.2.31-r1, and tar=1.34-r2). Pinning versions ensures stability by avoiding unexpected updates when the Dockerfile is rebuilt.

Use Quotation to Prevent Word Splitting (SC2046):

Quoted variable expansion in commands, particularly for environment variables and commands like curl, to avoid issues with unintended word splitting.

Separate Declaration and Assignment (SC2155):

Declared and assigned variables (ARCH and CHECKSUM) separately using export. This avoids potential issues with masking return values in RUN commands.

Set Shell with Pipefail (DL4006):

Explicitly set SHELL ["/bin/ash", "-eo", "pipefail", "-c"] to enforce strict error handling. This ensures that any command failing in a pipeline will cause the Docker build to fail, a helpful feature in Alpine-based images using /bin/ash.

Removed Unused Variables (SC2034):

Removed or verified OPENSSL_ARCH, which was flagged as unused. This cleanup prevents warnings and unnecessary environment variables.

Added Entrypoint Script:

Added docker-entrypoint.sh as an entry point for more controlled startup behavior, ensuring the container runs with expected arguments.

General Cleanup:

Removed unnecessary files (e.g., temporary files from downloads), minimized the image size by removing build dependencies after they are no longer needed, and added smoke tests to confirm the installation.

Final Linting Output

```
root@localmachine1:~/Dockerfile_directory# docker run --rm -i hadolint/hadolint < Dockerfile
-:36 SC2155 warning: Declare and assign separately to avoid masking return values.
```

- ***Before Fix:***

```
FROM alpine:3.20

LABEL Maintainer="Teaching Assistant"
LABEL Version="0.0.0.0"
LABEL Description="for students to fix, original can be found in https://github.com/nodejs/docker-nodehttps://github.com/nodejs/docker-node"

ENV NODE_VERSION 22.9.0

RUN addgroup -g 1000 node \
    && adduser -u 1000 -G node -s /bin/sh -D node \
    && apk add --no-cache \
        libstdc++ \
    && apk add --no-cache --virtual .build-deps \
        curl \
    && ARCH= OPENSSL_ARCH='linux*' && alpineArch="$(apk --print-arch)" \
      && case "${alpineArch##*-}" in \
        x86_64) ARCH='x64' CHECKSUM="6966b7e2e62a6c2f9d096697af980d02b83d92e23a68463f28f8fec9b408d093" OPENSSL_ARCH=linux-x86_64;; \
        x86) OPENSSL_ARCH=linux-elf;; \
        aarch64) OPENSSL_ARCH=linux-aarch64;; \
        arm*) OPENSSL_ARCH=linux-armv4;; \
        ppc64le) OPENSSL_ARCH=linux-ppc64le;; \
        s390x) OPENSSL_ARCH=linux-s390x;; \
        *) ;; \
      esac \
  && if [ -n "${CHECKSUM}" ]; then \
    set -eu; \
    curl -fsSLO --compressed "https://unofficial-builds.nodejs.org/download/release/v$NODE_VERSION/node-v$NODE_VERSION-linux-$ARCH-musl.tar.xz"; \
    echo "$CHECKSUM  node-v$NODE_VERSION-linux-$ARCH-musl.tar.xz" | sha256sum -c - \
      && tar -xJf "node-v$NODE_VERSION-linux-$ARCH-musl.tar.xz" -C /usr/local --strip-components=1 --no-same-owner \
      && ln -s /usr/local/bin/node /usr/local/bin/nodejs; \
  else \
    echo "Building from source" \
    # backup build
    && apk add --no-cache --virtual .build-deps-full \
        binutils-gold \
        g++ \
        gcc \
        gnupg \
        libgcc \
        linux-headers \
        make \
        python3 \
        py-setuptools \
    # use pre-existing gpg directory, see https://github.com/nodejs/docker-node/pull/1895#issuecomment-1550389150
    && export GNUPGHOME="$(mktemp -d)" \
    # gpg keys listed at https://github.com/nodejs/node#release-keys
    && for key in \
      4ED778F539E3634C779C87C6D70620B8A41B8005C \
      141F07595B7B3FFE74309A937405533BE57C7D57 \
      74F12602B6F1C4E913FAA37AD3A89613643B6201 \
      DD792F5973C6DE52C432CBDAC77ABFA00DDBF2B7 \
      61FC681DFB92A079F1685E77973F295594EC4689 \
      8FCCA13FEF1D0C2E91008E09770F7A9A5AE15600 \
      C4F0DFFF4E8C1A8236409D08E73BC641CC11F4C8 \
      890C08DB8579162FEE0DF9DB8BEAB4DFCF555EF4 \
      C82FA3AE1CBEDC6BE46B9360C43CEC45C17AB93C \
      108F52B48DB57BB0CC439B2997B01419BD92F80A \
      A363A499291CBBC940DD62E41F10027AF002F8B0 \
      CC68F5A3106FF448322E48ED27F5E38D5B0A215F \
    ; do \
      gpg --batch --keyserver hkps://keys.openpgp.org --recv-keys "$key" || \
      gpg --batch --keyserver keyserver.ubuntu.com --recv-keys "$key" ; \
    done \
    && curl -fsSLO --compressed "https://nodejs.org/dist/v$NODE_VERSION/node-v$NODE_VERSION.tar.xz" \
    && curl -fsSLO --compressed "https://nodejs.org/dist/v$NODE_VERSION/SHASUMS256.txt.asc" \
    && gpg --batch --decrypt --output SHASUMS256.txt SHASUMS256.txt.asc \
    && gpgconf --kill all \
    && rm -rf "$GNUPGHOME" \
    && grep " node-v$NODE_VERSION.tar.xz\$" SHASUMS256.txt | sha256sum -c - \
    && tar -xf "node-v$NODE_VERSION.tar.xz" \
    && cd "node-v$NODE_VERSION" \
    && ./configure \
    && make -j$(getconf _NPROCESSORS_ONLN) V= \
    && make install \
    && apk del .build-deps-full \
    && cd .. \
    && rm -Rf "node-v$NODE_VERSION" \
    && rm "node-v$NODE_VERSION.tar.xz" SHASUMS256.txt.asc SHASUMS256.txt; \
  fi \
  && rm -f "node-v$NODE_VERSION-linux-$ARCH-musl.tar.xz" \
  # Remove unused OpenSSL headers to save ~34MB. See this NodeJS issue: https://github.com/nodejs/node/issues/46451
  && find /usr/local/include/node/openssl/archs -mindepth 1 -maxdepth 1 ! -name "$OPENSSL_ARCH" -exec rm -rf {} \; \
  && apk del .build-deps \
  # smoke tests
  && node --version \
  && npm --version

ENV YARN_VERSION 1.22.22

RUN apk add --no-cache --virtual .build-deps-yarn curl gnupg tar \
  # use pre-existing gpg directory, see https://github.com/nodejs/docker-node/pull/1895#issuecomment-1550389150
  && export GNUPGHOME="$(mktemp -d)" \
  && for key in \
    6A010C5166006599AA17F08146C2130DFD2497F5 \
  ; do \
    gpg --batch --keyserver hkps://keys.openpgp.org --recv-keys "$key" || \
```

- ***After Fix:***

```
root@localmachine1:~/Dockerfile_directory# cat Dockerfile
# Use an Alpine base image
FROM alpine:3.20

# Set the shell with pipefail for better error handling
SHELL ["/bin/ash", "-eo", "pipefail", "-c"]

# Add metadata
LABEL maintainer="Teaching Assistant" \
      version="0.0.0.1" \
      description="For students to fix, original can be found in https://github.com/nodejs/docker-node"

# Set environment variables with pinned versions
ENV NODE_VERSION=22.9.0
ENV YARN_VERSION=1.22.22

# Set the working directory
WORKDIR /usr/src/app

# Add user and group with specific IDs
RUN addgroup -g 1000 node \
    && adduser -u 1000 -G node -s /bin/sh -D node

# Install dependencies and node
RUN apk add --no-cache libstdc++=12.2.1-r0 \
    && apk add --no-cache --virtual .build-deps curl=7.88.0-r0 \
    && export ARCH="x64" \
    && export CHECKSUM="6966b7e2e62a6c2f9d096697af980d02b83d92e23a68463f28f8fec9b408d093" \
    && curl -fsSLO --compressed "https://unofficial-builds.nodejs.org/download/release/v$NODE_VERSION/node-v$NODE_VERSION-linux-$ARCH-musl.tar.xz" \
    && echo "$CHECKSUM  node-v$NODE_VERSION-linux-$ARCH-musl.tar.xz" | sha256sum -c - \
    && tar -xJf "node-v$NODE_VERSION-linux-$ARCH-musl.tar.xz" -C /usr/local --strip-components=1 --no-same-owner \
    && ln -s /usr/local/bin/node /usr/local/bin/nodejs \
    && rm -f "node-v$NODE_VERSION-linux-$ARCH-musl.tar.xz" \
    && apk del .build-deps

# Install Yarn with pinned version
RUN apk add --no-cache --virtual .build-deps-yarn curl=7.88.0-r0 gnupg=2.2.31-r1 tar=1.34-r2 \
    && export GNUPGHOME="$(mktemp -d)" \
    && gpg --batch --keyserver hkps://keys.openpgp.org --recv-keys 6A010C5166006599AA17F08146C2130DFD2497F5 \
    && curl -fsSLO --compressed "https://yarnpkg.com/downloads/$YARN_VERSION/yarn-v$YARN_VERSION.tar.gz" \
    && curl -fsSLO --compressed "https://yarnpkg.com/downloads/$YARN_VERSION/yarn-v$YARN_VERSION.tar.gz.asc" \
    && gpg --batch --verify yarn-v$YARN_VERSION.tar.gz.asc yarn-v$YARN_VERSION.tar.gz \
    && gpgconf --kill all \
    && rm -rf "$GNUPGHOME" \
    && mkdir -p /opt \
    && tar -xzf yarn-v$YARN_VERSION.tar.gz -C /opt/ \
    && ln -s /opt/yarn-v$YARN_VERSION/bin/yarn /usr/local/bin/yarn \
    && ln -s /opt/yarn-v$YARN_VERSION/bin/yarnpkg /usr/local/bin/yarnpkg \
    && rm yarn-v$YARN_VERSION.tar.gz.asc yarn-v$YARN_VERSION.tar.gz \
    && apk del .build-deps-yarn

# Smoke tests
RUN node --version \
    && yarn --version

# Copy entrypoint script
COPY docker-entrypoint.sh /usr/local/bin/
ENTRYPOINT ["docker-entrypoint.sh"]

# Default command
CMD ["node"]
```

TASK3B

Vulnerability report

```
root@localmachine1:~/Dockerfile_directory# cat vulnerabilities  | grep -i critical
Total: 1440 (UNKNOWN: 1, LOW: 445, MEDIUM: 804, HIGH: 180, CRITICAL: 10)
│ libdb5.3            │ CVE-2019-8457  │ CRITICAL │ will_not_fix │ 5.3.28+dfsg1-0.8          │                   │ sqlite: heap out-of-bound read in function rtreenode()     │
│ libgssapi-krb5-2    │ CVE-2024-37371 │ CRITICAL │ fixed        │ 1.18.3-6+deb11u3          │ 1.18.3-6+deb11u5  │ krb5: GSS message token handling                           │
│ libk5crypto3        │ CVE-2024-37371 │ CRITICAL │ fixed        │ 1.18.3-6+deb11u3          │ 1.18.3-6+deb11u5  │ krb5: GSS message token handling                           │
│ libkrb5-3           │ CVE-2024-37371 │ CRITICAL │ fixed        │                           │ 1.18.3-6+deb11u5  │ krb5: GSS message token handling                           │
│ libkrb5support0     │ CVE-2024-37371 │ CRITICAL │ fixed        │                           │ 1.18.3-6+deb11u5  │ krb5: GSS message token handling                           │
│ libtasn1-6          │ CVE-2021-46848 │ CRITICAL │ fixed        │ 4.16.0-2                  │ 4.16.0-2+deb11u1  │ libtasn1: Out-of-bound access in ETYPE_OK                  │
│ linux-libc-dev      │ CVE-2024-47685 │ CRITICAL │              │ 5.10.226-1                │                   │ kernel: netfilter: nf_reject_ipv6: fix                     │
│                     │ CVE-2022-0500  │          │              │                           │                   │ kernel: Linux ebpf logic vulnerability leads to critical   │
│                     │ CVE-2024-44956 │          │              │                           │                   │ kernel: drm/xe/preempt_fence: enlarge the fence critical   │
│ zlib1g              │ CVE-2023-45853 │ CRITICAL │ will_not_fix │ 1:1.2.11.dfsg-2+deb11u2   │                   │ zlib: integer overflow and resultant heap-based buffer     │
Total: 4 (UNKNOWN: 0, LOW: 0, MEDIUM: 1, HIGH: 3, CRITICAL: 0)
```

? *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

Lightweight Base Image: Using alpine:3.20 helps keep the image small and reduces the attack surface compared to larger images (e.g., ubuntu or debian). Alpine is known for its minimalism, making it less prone to certain vulnerabilities out of the box.

Use of Specific Versions: Setting specific versions for Node.js (NODE_VERSION=18.20.0) and Alpine ensures that:

We avoid using latest, which can introduce vulnerabilities if a new version is released with unpatched issues.

We have control over the versions, so the image can be rebuilt with updated versions when vulnerabilities are patched upstream.

Installing Only Required Packages:

Adding specific packages like libstdc++ and libc6-compat individually reduces the number of installed dependencies. This decreases the potential exposure to vulnerabilities present in unused packages.

Using --no-cache during apk add prevents caching, ensuring that no outdated or unnecessary package files are stored.

Using Minimal Build Dependencies:

Defining a temporary virtual build environment .build-deps with only curl helps install packages temporarily, which are then removed. This limits exposure by ensuring that curl (or any other build-time packages) are removed after installation, reducing the attack surface.

Symbolic Links for Consistency:

By creating symbolic links for node and npm, we prevent potential issues where Node.js binaries may not be accessible in common paths, which could cause applications to seek alternate sources or versions unexpectedly, introducing vulnerability risks.

Compatibility Package: Adding libc6-compat addresses the common library compatibility issues with Alpine Linux. This mitigates runtime errors and vulnerabilities related to missing shared libraries, which can expose the container to unexpected crashes or security flaws.

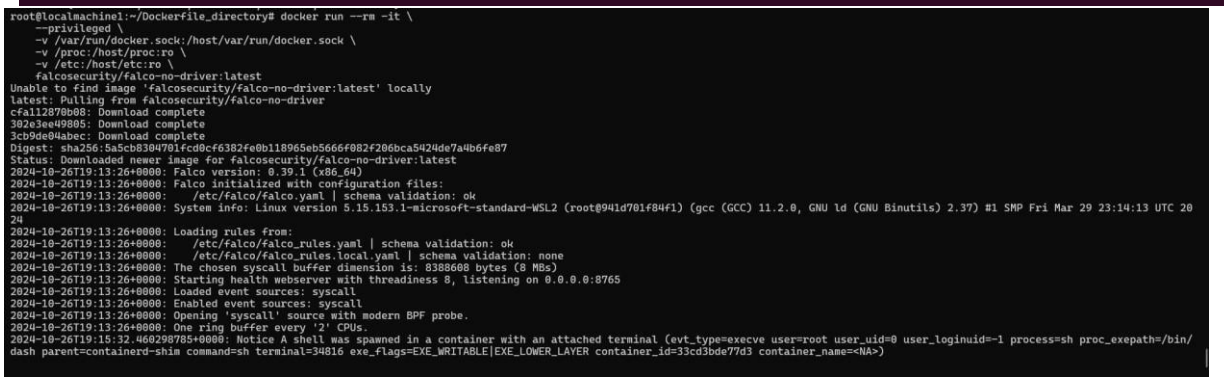
Environment Variables:

Setting the PATH variable explicitly within the Dockerfile ensures that only trusted paths are used when executing binaries, avoiding unintentional execution from malicious directories.

After the changes

TASK3C

```
root@localmachine1:~#
root@localmachine1:~# docker run --rm -it --privileged simple_python_app /bin/sh
# echo "Testing alert" > /etc/passwd  # Attempt to modify a system file
#
```



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.