Container Security

Cybersecurity

Supplemental LP

Class Objectives

By the end of class, you will be able to:

- Explain the usage of a container vulnerability scanning tool before deploying containers and using container intrusion detection systems to monitor them once running.
- Use Trivy to scan a container image and retrieve its results from the JSON output.
- Use the bash command-line tool jq to parse through JSON.
- Monitor container intrusion detection systems for anomalies.

In past lessons, we've seen how containers can allow us to deploy multi-component web applications that separates each of the application's components by its individual concern.

In today's activities, we will be revisiting containers and looking at some of the daily tasks a security professional does to help an organization secure its containerized infrastructure.

What Goes Into a Container?

```
##############################################
## An Insecure and Outdated Container Build
##############################################
## Required Base OS
FROM ubuntu:14.04
RUN apt-get update -y && apt-get upgrade -y
WORKDIR /home/user/Downloads
## Install wget from the Debian repo
RUN curl
http://archive.ubuntu.com/ubuntu/pool/main/w/wget/wget_1.15-1ubuntu1.14.04.5_amd64.deb -O
&& \
    dpkg -i wget_1.15-1ubuntu1.14.04.5_amd64.deb
## Install bash 4.3
RUN curl https://ftp.heanet.ie/mirrors/gnu/bash/bash-4.3.tar.gz -O && \
    tar zxf bash-4.3.tar.gz && \
    mkdir /opt/bash && cd bash-4.3 && \
    ./configure --prefix=/opt/bash && \
    make && make install
## Copy application files
COPY . /app
## Change working directory
WORKDIR /app
## Launch application from /app
RUN deploy.sh
```

```
##############################################
## An Insecure and Outdated Container Build
##############################################
## Required Base OS
FROM ubuntu:14.04
RUN apt-get update -y && apt-get upgrade -y
WORKDIR /home/user/Downloads
## Install wget from the Debian repo
RUN curl
http://archive.ubuntu.com/ubuntu/pool/main/w                    .deb -O
&& \
    dpkg -i wget_1.15-1ubuntu1.14.04.5_amd64
## Install bash 4.3
RUN curl https://ftp.heanet.ie/mirrors/gnu/bash/bash-4.3.tar.gz -O && \
    tar zxf bash-4.3.tar.gz && \
    mkdir /opt/bash && cd bash-4.3 && \
    ./configure --prefix=/opt/bash && \
    make && make install
## Copy application files
COPY . /app
## Change working directory
WORKDIR /app
## Launch application from /app
RUN deploy.sh
```

Indicates that Ubuntu 14.04 is the base operating system of the application. This means that all security issues that are present in Ubuntu version 14.04 also exist in this container.

```
##############################################
## An Insecure and Outdated Container Build
##############################################
## Required Base OS
FROM ubuntu:14.04
RUN apt-get update -y && apt-get upgrade -y
WORKDIR /home/user/Downloads
## Install wget from the Debian repo
RUN curl
http://archive.ubuntu.com/ubuntu/pool/main/w/wget/wget_1.15-1ubuntu1.14.04.5_amd64.deb -O
&& \
    dpkg -i wget_1.15-1ubuntu1.14.04.5_amd64.deb
## Install bash 4.3
RUN curl https://ftp.heanet.ie/mirrors/gnu/bash/bash-4.3.tar.gz -O && \
    tar zxf bash-4.3.tar.gz && \
    mkdir /opt/bash && cd bash-4.3 && \
    ./configure --prefix=/opt/bash && \
    make && make install
## Copy application files
COPY . /app
## Change working directory
WORKDIR /app
## Launch application from /app
RUN deploy.sh
```

In the RUN lines, two vulnerable packages, wget 1.15 and bash 4.3, get installed.

```
##############################################
## An Insecure and Outdated Container Build
##############################################
## Required Base OS
FROM ubuntu:14.04
RUN apt-get update -y && apt-get upgrade -y
WORKDIR /home/user/Downloads
## Install wget from the Debian repo
RUN curl
http://archive.ubuntu.com/ubuntu/pool/main/w/wget/wget_1.15-1ubuntu1.14.04.5_amd64.deb -O
&& \
    dpkg -i wget_1.15-1ubuntu1.14.04.5_amd64.deb
## Install bash 4.3
RUN curl https://ftp.heanet.ie/mirrors/gnu/bash/bash-4.3.tar.gz -O && \
    tar zxf bash-4.3.tar.gz && \
    mkdir /opt/bash && cd bash-4.3 && \
    ./configure --prefix=/opt/bash && \
    make && make install
## Copy application files
COPY . /app
## Change working directory
WORKDIR /app
## Launch application from /app
RUN deploy.sh
```

Lastly, a script deploy.sh is run to start up this container's application.

Vulnerable Containers

The wget version was vulnerable to reverse directory traversals and code execution and the bash version was vulnerable to Shellshock.

As security engineers, we need to be able to inspect a container's operating system and its installed software packages for vulnerabilities.

Vulnerability Scanning with Trivy

Outdated software packages can lead to an application being vulnerable to the following vulnerabilities:

- **Man-in-the-middle (MITM):** An attacker secretly intercepts and relays requests between a victim and their intended target, such as a website.
- **Buffer overflow:** A software anomaly in which the memory of an application is corrupted and foreign code is executed as a part of memory.
- **Denial of service (DoS)**: Takes down a service and makes it unavailable.
- **Privilege escalation:** Enables one to escape the regular user space of an operating system to run commands with elevated privileges

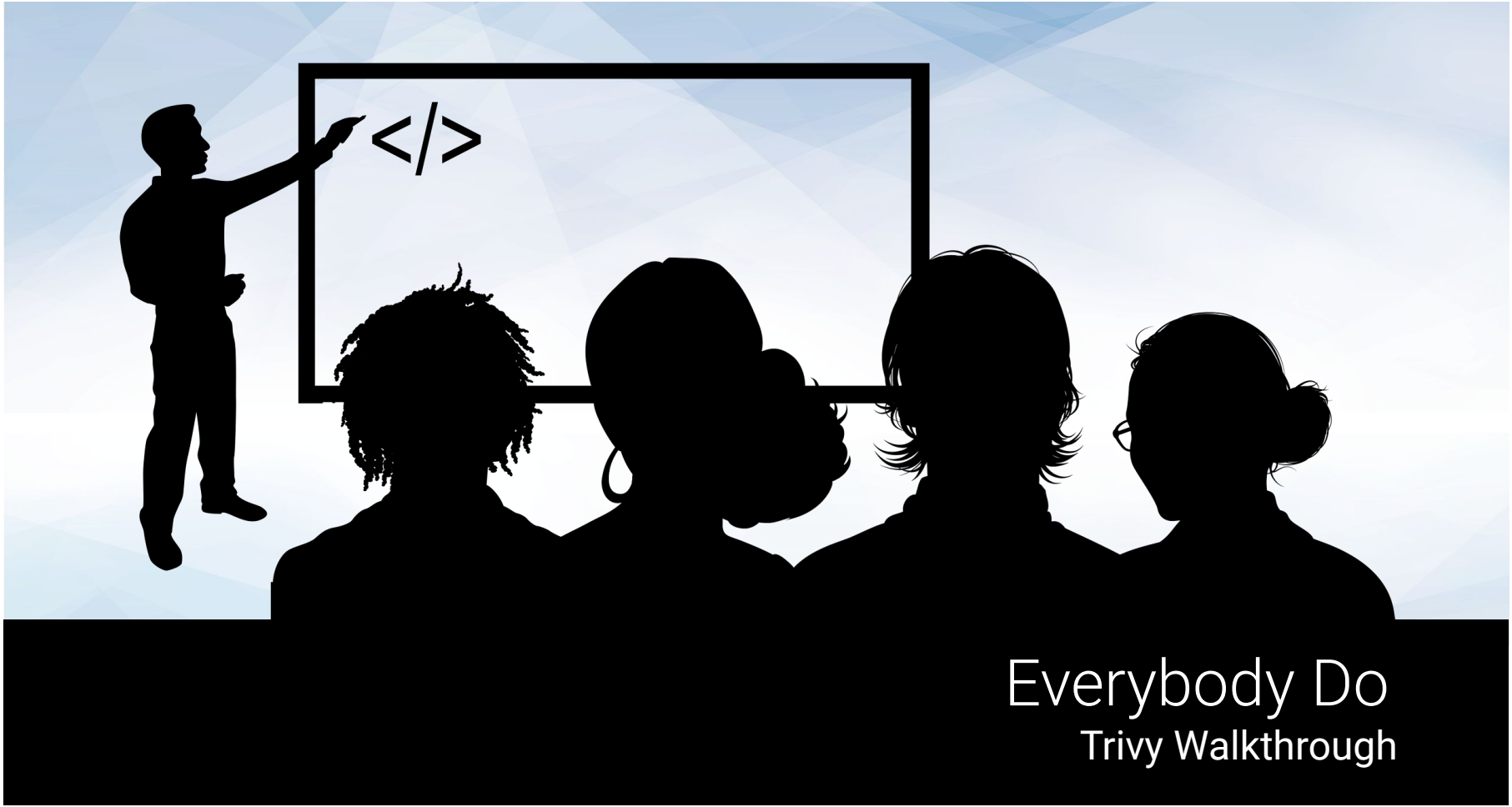
Scanning container images and Dockerfiles before they are deployed is called **vulnerability scanning**.

It is one of the earliest stages in the application deployment process that security engineers become involved.

Walkthrough Scenario

In the next, walkthrough, we will use the vulnerability scanner Trivy

- We will play the role of a web app security engineer as part of the company's Systems Engineer and Development team
- The developers insist on using the following Docker image for the general microservice architecture planned by your company's systems architect: wordpress:4.6.1-php5.6-apache.
- However, the senior security engineer insists that using such an old Docker image of WordPress presents a significant security risk.
- We are tasked with using Trivy to scan the image, generate a comprehensive list of the most severe vulnerabilities, and report our findings to the Risk Management team.


Everybody Do
Trivy Walkthrough

Bash Filtering

Now that we know how to use Trivy, we'll filter the results using bash command-line tools.

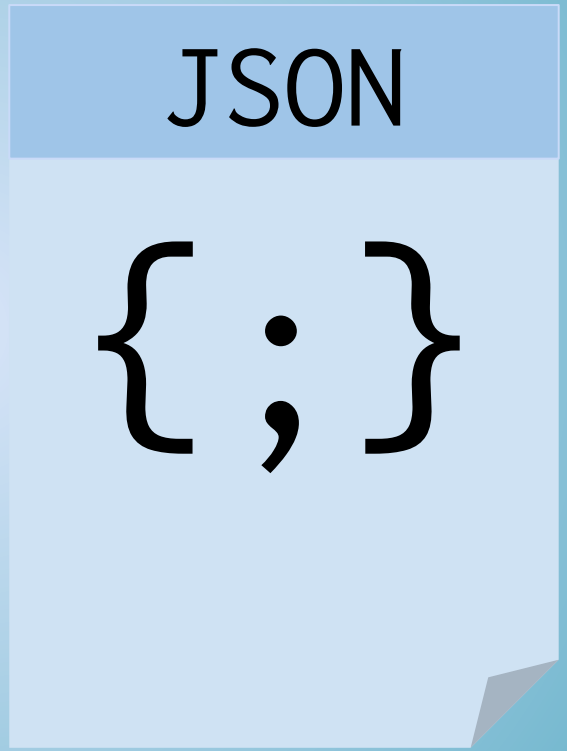
We will need to use the command-line tool **`jq`** to inspect the **`results.json`** file that we created with Trivy.

- **`jq`** is like **`sed`**, but for JSON data. It's used to filter through JSON files and has the same text editing functionalities built into **`sed`**, **`awk`**, and **`grep`**.
- JSON is a structured data format that is used often in software development languages, logging, and security.

JSON

While we won't need to know JSON in depth, we'll need to know how to interact with it using `jq`, to filter through vulnerability reports.

JavaScript Object Notation is a language-agnostic data format used for web services and APIs, as well as log aggregators such as ELK and Splunk.



JSON

A high level overview of JSON formatting:

```
{
"thisIsAKey":"this is a value",
"theFollowingIs":"an_array_example",
"ciaTriad":[ "confidentiality", "integrity", "availability" ]
}
```

- It uses **`key:value`** pairs.
- Data is separated by commas.
- Curly braces can hold objects.
- Square brackets hold arrays.

JSON

A high-level overview of **`jq`** syntax. This searches through the **`Vulnerabilities`** in the first array (or listed data) of the **`results.json`** file.

```
jq '.[0].Vulnerabilities' results.json
```

- **`VulnerabilityID`**: The Common Vulnerabilities and Exposures ID (CVD), a unique ID given to every vulnerability.
- **`PkgName`**: contains the package name that is vulnerable.
- **`InstallVersion`**: Version of the vulnerable package.
- **`FixedVersion`**: Fixed version of the package.
- **`Title:`** Title of the vulnerability.
- **`Description:`** Description of the vulnerability.
- **`Severity:`** The level of potential risk to the system it is installed on.
- **`References`**: Links to more information.

JSON

We can use pipes with our **`jq`** command to filter for specific results.

```
jq '.[0].Vulnerabilities[] | select(.Description | test("Man-in-The-Middle"))'
                    results.json > results_mitm_vulns.json
```

- This command will search the for the phrase “Man-in-the-Middle” in the description section of the vulnerabilities list of the **`result.json file`**.
- It will then send those results to a new file named **`results_mitm_vulns.json.`**

Everybody Do

Trivy Walkthrough Continued

Trivy Wrap Up

Everyone should now have a `~/Documents/trivy_reports` directory containing the following:

- A full wordpress:4.6.1-php5.6-apache container image vulnerability listing: wp_all_vulns.json
- A filtered buffer overflow vulnerability list: wp_bo_vulns.json
- A filtered denial of service vulnerability list: wp_dos_vulns.json
- A filtered privilege escalation vulnerability list: wp_pe_vulns.json

```
sysadmin@UbuntuDesktop:~/Documents/trivy_reports$ ls -l
    total 1788
    -rw-r--r-- 1 sysadmin sysadmin 1291723 Nov 13 03:21 wp_all_vulns.json
    -rw-r--r-- 1 sysadmin sysadmin  161488 Nov 13 04:00 wp_bo_vulns.json
    -rw-r--r-- 1 sysadmin sysadmin  325121 Nov 13 04:10 wp_dos_vulns.json
    -rw-r--r-- 1 sysadmin sysadmin   41059 Nov 13 04:10 wp_pe_vulns.json
```


Break

Container Runtime IDS

Now we’ll focus on another container-related security task: monitoring.

IDS Review

We covered IDS in out network security unit:

An intrusion is any unwanted malicious activity within a network or a system.

An IDS is a device or software application that monitors a network or system for malicious activity.

A host-based intrusion detection system (HIDS) detects intrusion attempts on an endpoint device.

A network intrusion detection system (NIDS) detects intrusion attempts within a network.

Container Intrusion Detection Systems (CIDS)

Containers have the same vulnerabilities as underlying VMs. But the large-scale deployment of containers vastly multiplies their attack surface.

Even though we scanned container Dockerfiles for vulnerabilities, we haven't done anything to secure the containers once they are deployed.

After container sets are deployed with tools like Docker Compose in live environments, we need to be able to detect intrusions.

Similar to NIDS and HIDS, container intrusion detection systems (CIDS) detect intrusions specifically for deployed containers.

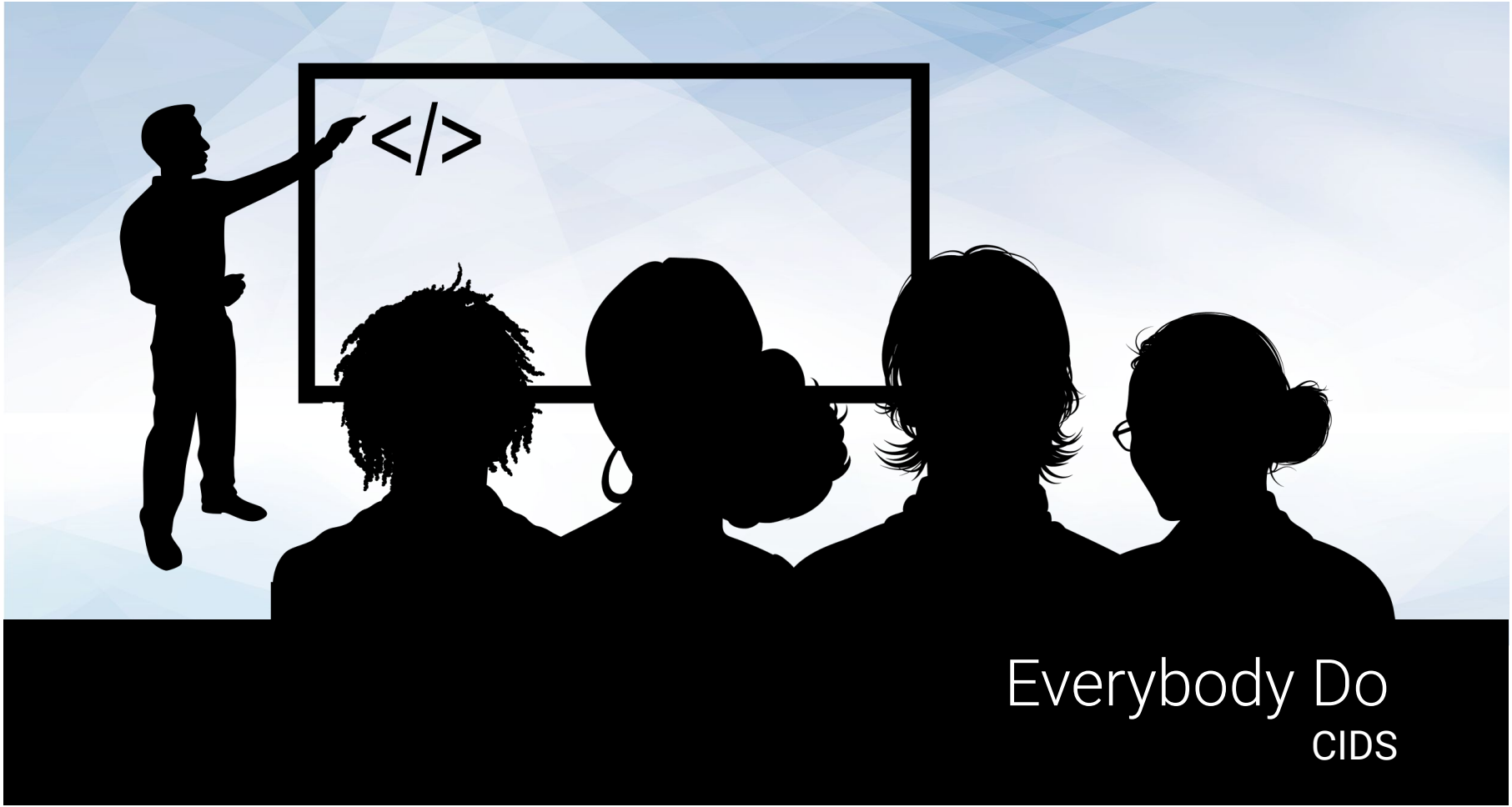
The specific CIDS we will use today is called Falco.

Falco is an open-source CIDS that alerts security professionals to potential intrusion attempts.

Common intrusion behavior on containers include:

- **New shell running inside a container**. An unknown shell starting in a container signals an attacker has potentially accessed a system and will start executing commands.
- **When a sensitive file, such as `/etc/shadow`, is read**. At no point during a container deployment should the contents of **`/etc/shadow`** be read. This is a clear sign that an attacker is retrieving the hashed passwords of accounts within the system.
- **Unprompted configuration changes**. After an attacker gains access to a system, they will often create a user to maintain persistence within that system.
- **File creations at `/root` or `/`.** At no point during a container's runtime should new files be created in the **`/root`** or **`/`** directories. Files created in these directories indicate compromise.

In the upcoming walkthrough, we will use a multi-container Docker Compose deployment in conjunction with Falco while simulating various types of intrusion activity.


Everybody Do
CIDS


The End.