Module 6:
SCA

Software Composition Analysis (SCA)

- Software Composition Analysis (SCA) tools
- These are designed to help identify and manage **open-source and third-party components within software applications**. These tools are crucial for ensuring the security and compliance of software projects by detecting and managing vulnerabilities in the **third-party libraries** and components that developers use.
- Software is rarely developed in isolation or rather software is often developed with the help of **third party dependencies**. Whether these be frameworks or dependencies that have functionality you can leverage instead of writing yourself.
- Can result in a dependency tree or dependency graph
- We are trying to determine all of the used dependencies and whether or not they are on up to date versions or are vulnerable to any CVEs

SCA

- It is realistically a form of SAST, as it is testing the source code where the dependencies are managed
- Similarly well suited for CI/CD pipelines (SAST)
- It will test all of the components that make up the software for known vulnerabilities. These generally tend to work best with open source components
- It is these dependencies that can be flagged by SCA tools as being out of date and vulnerable.
- For example, you might be using **version 1** of a **JavaScript library** which provides common functions and an SCA tool might flag that this version is vulnerable to multiple CVEs and should be upgraded to the latest version

SCA

Here are some popular SCA tools:

- OWASP Dependency Check
- Snyk
- White Source
- Black Duck
- Checkmarx

OWASP Dependency Check

- OWASP stands for the **Open Web Application Security Project**
- It is a nonprofit organization focused on improving the security of software.
- OWASP is known for its efforts to provide freely available security-related resources for organizations and individuals.
- The organization operates as a community of security experts and practitioners who collaborate to create tools, documentation, and standards to help organizations develop and maintain secure software.
- One of OWASP's most well-known contributions is the **OWASP Top Ten**, a list that outlines the ten most critical web application security risks. This list is periodically updated to reflect the evolving landscape of security threats.

OWASP Dependency Check

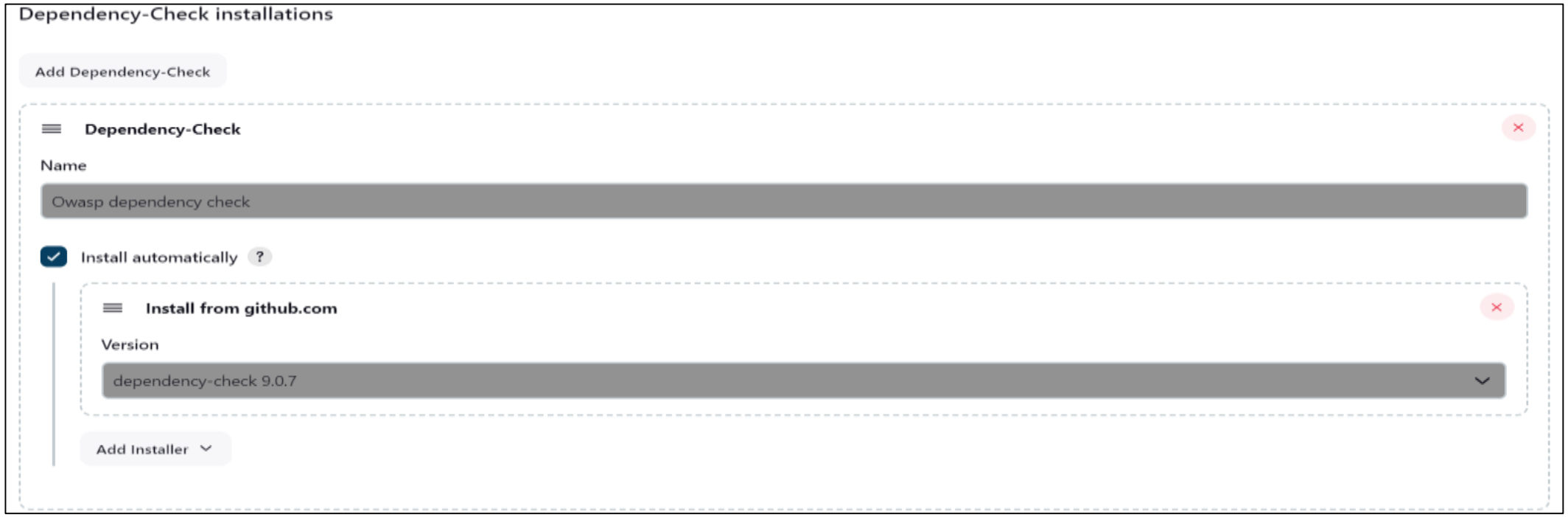
Dependency-Check is available as a:

- Command-line utility
- Ant Task
- Gradle Plugin
- Jenkins Plugin
- Maven Plugin
- SonarQube Plugin

Integration with Jenkins

1. Install OWASP Dependency-check plugin (Manage Jenkins → Plugins → Available Plugins)

2. Configure tool configuration of Dependency-check (Manage Jenkins → tools → Dependency-check installations → add Dependency-check)



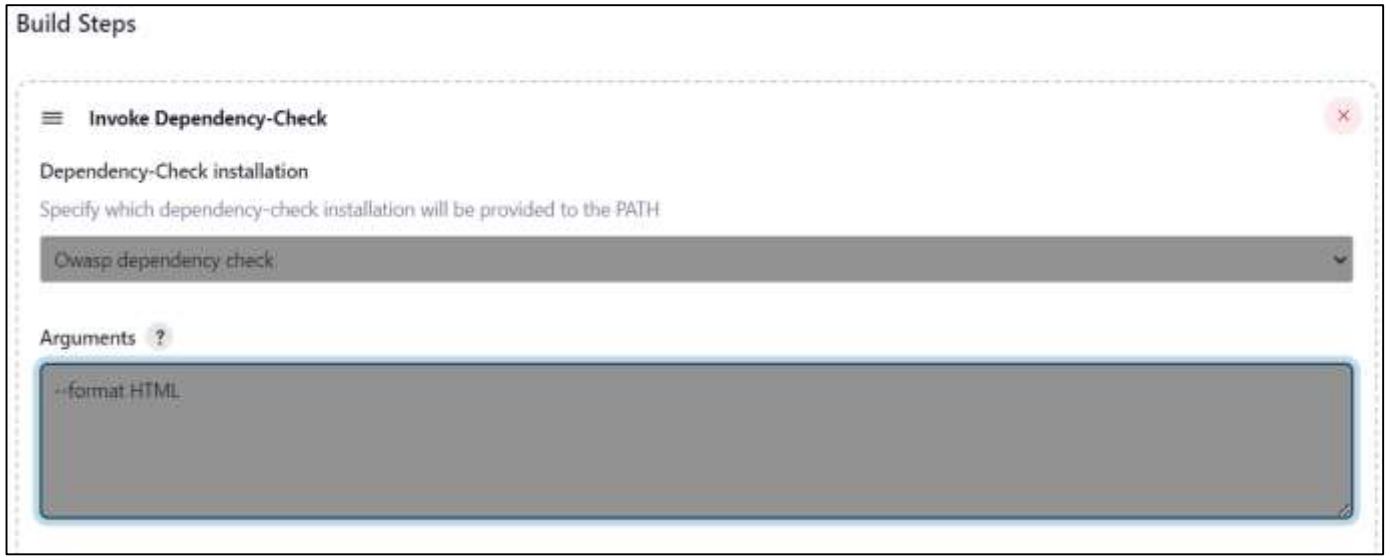
Integration with Jenkins

3. Create or configure a Freestyle Project Jenkins.

4. Select “invoke Dependency check” under build steps

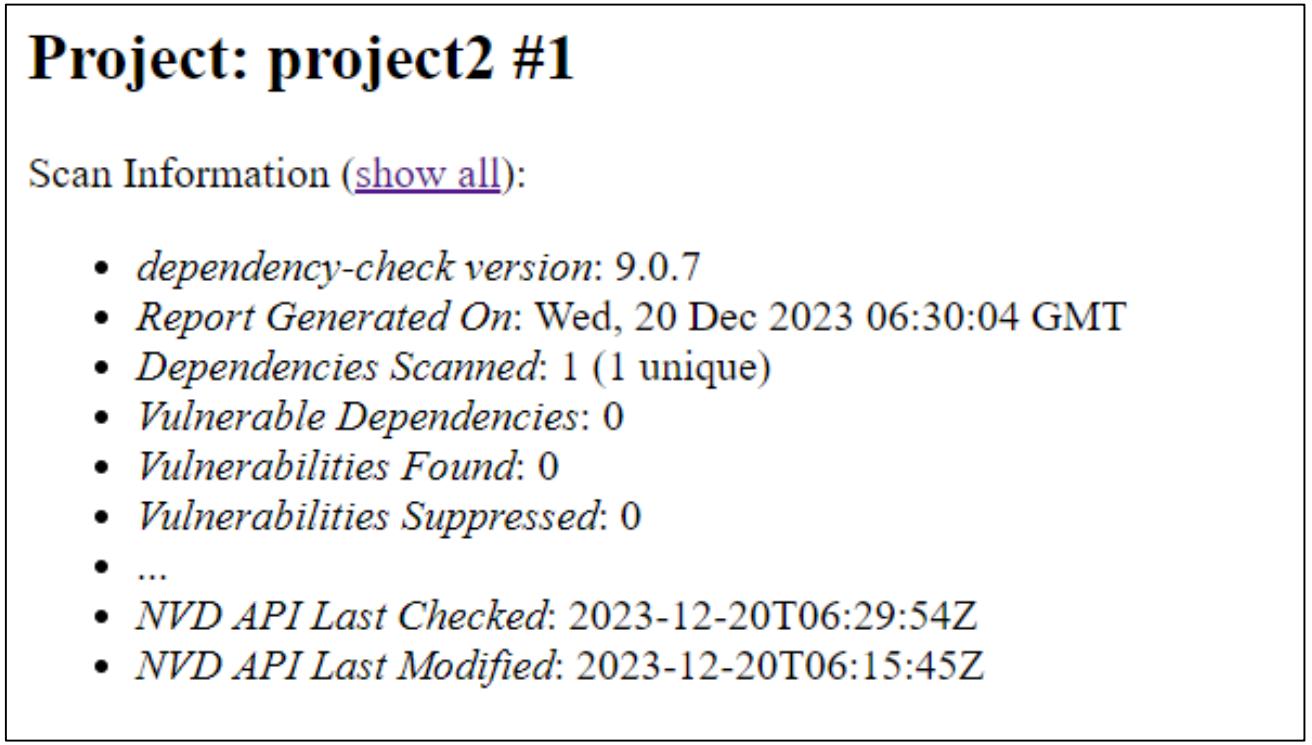
5. Use the arguments as per the requirements.

https://jeremylong.github.io/DependencyCheck/dependency-check-cli/arguments.html



Integration with Jenkins

6. Check the status (workspace → dependency-check-report.html)



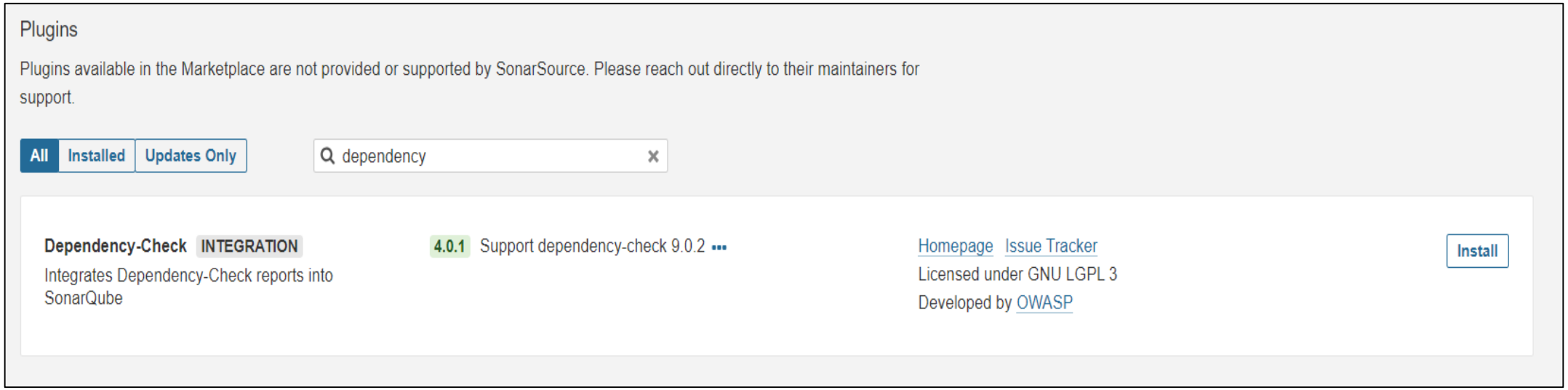
Project: project2 #1

Scan Information (show all):

- *dependency-check version*: 9.0.7
- *Report Generated On*: Wed, 20 Dec 2023 06:30:04 GMT
- *Dependencies Scanned*: 1 (1 unique)
- *Vulnerable Dependencies*: 0
- *Vulnerabilities Found*: 0
- *Vulnerabilities Suppressed*: 0
- ...
- *NVD API Last Checked*: 2023-12-20T06:29:54Z
- *NVD API Last Modified*: 2023-12-20T06:15:45Z

Dependency-check plugin in SonarQube

Install the plugin dependency-check (Go to Marketplace)



Module 7:
DAST
Dynamic Application Security Testing

DAST

- Dynamic application security testing
- **Black box testing** (it can not see everything behind the scenes)
- It can test a running application through its UI or API for common vulnerabilities such as SQL injection and buffer overflows.
- Security testing is done from the outside
- Simulating the attacks to **penetrate the application** to look for possible **vulnerabilities** and **flaws**
- Performed in the later stages of SDLC.
- DAST is performed during the testing phase to check if there are any security flaws that are left behind during development phase
- It can be embedded into CI/CD pipelines, but it would be for completeness and are aimed at manual testing as a part of a dedicated penetration test.

DAST


Plan
Threat landscape, change impact analysis
Release
Access and configuration management
Code
Pre-commit hooks, SAST
Deploy
Chaos engineering, pen test
Build
Software component analysis, SAST
Monitor
Log collection, SIEM, RASP
Test
Authentication, SQL injection, DAST
Respond
Block attacks, roll back
1
2
3
4
5
6
7
8
Pre-production
Production

Web Application Security best practices

- Require input validation
- Data encryption
- Authentication and access control
- Avoid security misconfigurations

SAST vs DAST

SAST	DAST
White box testing	Black box testing
Source code is required	Requires a running web application
Earlier stages of SDLC	Later stages of SDLC

DAST Tools

Tool	Language Support	Description
Valgrind	C, C++	A widely used open-source tool for memory management and debugging in C and C++ applications.
Apache JMeter	Java	A popular tool for load testing and performance measurement of web applications and services.
Apache Bench (ab)	Apache HTTP Server (C, Perl)	A command-line tool for benchmarking the performance of web servers and applications.
GDB (GNU Debugger)	C, C++, Python, Ada, more	A versatile and powerful debugger with dynamic analysis capabilities for various programming languages.
Burp Suite	Various (primarily Java)	A security testing tool for web applications, providing dynamic scanning and vulnerability assessment.

DAST Tools

- Commercial – Netsparker, Acunetix
- Open-source – OWASP ZAP

OWASP

- Non profit foundation with the aim of improving the security of software
- They produce multiple projects which help with specific areas of security e.g. **tooling, testing guides and vulnerable software** (to test against) and many more

OWASP

Projects:

- **OWASP ZAP** – A penetration testing tool
- **OWASP Top 10** – A document outlining the top most critical security concerns for web application security
- **OWASP ASVS** – a list of requirements for secure application development
- **OWASP Cheat sheets** – Simple good practice guides for application security
- **OWASP ModSecurity Core Rule Set** – A set of rules to be used with the ModSecurity Web application firewall (rather than pay for an enterprise alternative)
- **OWASP Secure Headers Project** – A project that describes what HTTP headers you should set in your web application.

Introducing ZAP

- **Zed Attack Proxy (ZAP)** is a free, open-source penetration testing tool being maintained under the umbrella of The Software Security Project (SSP).
- ZAP is designed specifically for testing web applications.
- ZAP is what is known as a "**man-in-the-middle proxy**."
- It stands between the tester's browser and the web application so that it can intercept and inspect messages sent between browser and web application, modify the contents if needed, and then forward those packets on to the destination.
- It can be used as a stand-alone application, and as a daemon process.

What is ZAP?

- Zed Attack Proxy
- A tool for penetration test
- A tool for finding vulnerability in web applications
- An OWASP flagship Project
- Free and Open Source
- Cross Platform
- Well Maintained
- Probably the worlds most frequently used web scanner

Automation Options

- Command Line
- Jenkins Plugin
- Packaged Scans
- Github Actions
- Daemon + API

ZAP- Baseline Scan

- A **ZAP Baseline Scan** refers to a basic security scan performed using the **OWASP Zed Attack Proxy (ZAP)** to identify common vulnerabilities in a web application.
- The goal of a baseline scan is to quickly assess the security posture of an application by **highlighting potential issues** without going into in-depth analysis.
- The baseline scan is a starting point for further, more detailed security testing.

Command : **zap-baseline.py -t <target>** [options]

-t target target URL including the protocol, eg **https://www.example.com**

OWASP TOP 10

- A document containing the 10 most critical security concerns for web application security
- Released every few years (historical releases have been 2003, 2004, 2007, 2010, 2013, 2017, 2021)
- Historically released as a PDF

Major Web application Security Threats

1. Broken Access Control

If **authentication and access restriction** are not properly implemented, it's easy for attackers to take whatever they want. With broken access control flaws, **unauthenticated or unauthorized** users may have access to **sensitive files and systems**, or even user privilege settings.

Penetration testing can detect missing authentication but cannot determine the misconfigurations that lead to the exposure. One of the benefits of the increasing use of Infrastructure as Code (IaC) tools is the ability to use scanning tools to detect configuration errors leading to access control failures.

Weak access controls and issues with credentials management in applications are preventable with secure coding practices, as well as preventative measures like locking down administrative accounts and controls and using multi-factor authentication.

Major Web application Security Threats

2. Cryptographic Failure
Common errors such as using hardcoded passwords, outdated cryptographic algorithms, or weak cryptographic keys can result in the exposure of sensitive data Scanning images for hardcoded secrets, and ensuring that data is properly encrypted at rest and in transit can help mitigate exposing sensitive data to attackers.

Major Web application Security Threats

3. Injection
Injection attacks occur when attackers exploit vulnerabilities in web applications that **accept untrusted data**. Common types include **SQL injection** and **OS command injection**. This category now also includes **Cross Site Scripting** (XSS). By inserting malicious code into input fields, attackers can execute unauthorized commands, access sensitive databases, and potentially gain control over systems. Application security testing can reveal injection flaws and suggest remediation techniques such as stripping special characters from user input or writing parameterized SQL queries.

Major Web application Security Threats

4. Insecure Design

Insecure design is a new category in the 2021 OWASP Top Ten which focusses on fundamental design flaws and ineffective controls as opposed to weak or flawed implementations.

Creating secure designs and secure software development lifecycles requires a combination of culture, methodologies and tools. Developer training, robust **threat modelling**, and an organizational library of secure design patterns should all be implemented to reduce the risks of insecure designs creating critical vulnerabilities.

Major Web application Security Threats

5. Security Misconfiguration

Application servers, frameworks, and cloud infrastructure are highly configurable, and security misconfigurations such as too broad permissions, insecure default values left unchanged, or too revealing error messages can provide attackers easy paths to compromise applications.

The 2023 Veracode State of Software Security reported that misconfiguration errors were reported in 70% or more applications that had introduced a new vulnerability in the last year.

To reduce misconfiguration risks organizations should routinely harden deployed application and infrastructure configurations and should scan all infrastructure as code components as part of a secure SDLC.

Major Web application Security Threats

6. Vulnerable and Outdated Components

Modern applications are built using a large number of **third-party libraries** (which themselves are dependent on other libraries), and frequently run on open-source frameworks. In a modern application there may be orders of magnitude more code from libraries and components than written by an organization's developers.

As might be expected with any software, vulnerabilities in libraries and components will routinely be discovered, patched, and new versions released. The challenges of identifying all the components in use, keeping track of their vulnerability status, and routinely rebuilding and testing deployed software is both essential and onerous. Perhaps this is why so many organizations are still running vulnerable software in production.

Major Web application Security Threats

7. Identification and Authentication Failures
Identifying and authorizing users and non-human clients is a fundamental security practice. It goes without saying that weaknesses in a way an application allows access or identifies users is a critical vulnerability. While mitigation starts with secure coding practices, tools to detect and prevent credential stuffing and brute force attacks are also useful protections.

Major Web application Security Threats

8. Software and Data Integrity Failures

The tools used to build, manage, or deploy software are increasingly common vectors of attack. A CI/CD pipeline that routinely builds, tests and deploys software can also be used to inject malicious code (or libraires), create insecure deployments, or steal secrets.

As discussed above in ‘Vulnerable and Outdated Components’ modern applications use many third-party components, often pulling them from third party repositories.

Organizations can mitigate this threat by ensuring both the security of the build process, and the components pulled into the build. Adding in code scanning and software component analysis steps into a software build pipeline can identify malicious code or libraries. Ensuring the build and

Major Web application Security Threats

9. Security Logging and Monitoring Failures

Having adequate logging and monitoring in place is essential in both detecting a breach early, hopefully limiting the damage, and in incident forensics to establish the scope of the breach, and to determine the method of compromise.

Simply generating the data is obviously insufficient, organizations must have adequate collection, storage, alerting and escalation processes. Organizations should also verify that these processes are working correctly – using Dynamic Application Security Testing (DAST) tools like Veracode DAST, for instance, should produce significant logging and alerting events.

Major Web application Security Threats

10. Server-Side Request Forgery (SSRF)

Modern web applications commonly fetch additional content or data from a remote resource. If an attacker can influence the destination resource, and the application does not validate the supplied URL, then a crafted request may be sent to a target destination.

Mitigating SSRF attacks is done using familiar methods such as sanitizing user input, using explicit allow lists, and inspecting request responses before they are returned to clients.

OWASP ASVS

- The OWASP **Application Security Verification Standard (ASVS)** Project provides a basis for testing web application technical security controls and also provides developers with a list of requirements for secure development.
- It can be used as a checklist when testing web applications. For Example if you were testing a logon page you can reference the ASVS and see exactly how the logon page should behave vs how it does behave.
- Provided in multiple formats e.g PDF, Word, CSV
- https://github.com/OWASP/ASVS/tree/v4.0.3?tab=readme-ov-file#latest-stable-version---403

OWASP Cheat Sheets

- Concise information on application security topics e.g. what to do and what not to do
- For example, when it comes to authentication this is a huge topic that could be daunting to first try and understand. The authentication cheat sheet is concise and breaks down exactly where to focus.
- Cheat sheets also exist for the OWASP Top 10 and OWASP ASVS which index which cheat sheets are relevant so can be directly correlated between the two.
- https://cheatsheetseries.owasp.org/

CIS Benchmarks

- Produced by CIS (**Centre for internet security**)
- Non profit with the aim of securing organisations against **cyber threats**
- CIS benchmarks are **configuration guidelines** for various products e.g **software, operating systems, network devices, mobile devices** etc
- Be aware as they can make your life easier e.g. need to ensure a server is hardened? Check against specific OS's CIS benchmark
- They also provide hardened images e.g. out of the box for Azure, making your life easier as they are pre-hardened
- Some tools will test against benchmarks

CIS Controls

- Formerly known as **SANS Top 20**
- Currently a list of 18 critical security controls
- They are a general set of recommended best practices for securing an organisation and its data.

CIS Controls

CIS Control 1: Inventory and Control of Enterprise Assets

CIS Control 2: Inventory and Control of Software Assets

CIS Control 3: Data Protection

CIS Control 4: Secure Configuration of Enterprise Assets and Software

CIS Control 5: Account Management

CIS Control 6: Access Control Management

CIS Control 7: Continuous Vulnerability Management

CIS Control 8: Audit Log Management

CIS Control 9: Email and Web Browser Protections

CIS Controls

CIS Control 10: Malware Defenses

CIS Control 11: Data Recovery

CIS Control 12: Network Infrastructure Management

CIS Control 13: Network Monitoring and Defense

CIS Control 14: Security Awareness and Skills Training

CIS Control 15: Service Provider Management

CIS Control 16: Application Software Security

CIS Control 17: Incident Response Management

CIS Control 18: Penetration Testing

Introduction to Snyk

Snyk is a developer security platform that enables application and cloud developers to secure their whole application — finding and fixing vulnerabilities from their first lines of code to their running cloud.

It is helping organizations and developers secure their software applications throughout the development lifecycle.

Snyk provides a range of tools and services designed to identify and mitigate security vulnerabilities and issues in software, particularly in the context of open-source dependencies and libraries.

Brief History and Background

- Founding and Early Development (2015-2016)
- Launch and Growth (2016-2018)
- Container and Serverless Security (2018-2019)
- Funding and Investment (2015-2020)
- Widespread Adoption and Community Involvement (2020-Present)

Impact of Security Breaches

- Financial Losses
- Reputation Damage
- Loss of Trust
- Legal and Regulatory Consequences
- Operation Disruptions
- Data Loss and Privacy Concerns
- Loss of Competitive Edge
- Crisis Response Costs

Key Features of Snyk

Vulnerability Scanning:

- Snyk scans an organization's codebase, including open source and proprietary code, to identify vulnerabilities and security issues.

Container Security:

- Snyk assesses container images for vulnerabilities and misconfigurations, helping organizations ensure the security of their containerized applications.

Infrastructure as Code (IaC) Security:

- Snyk can scan Infrastructure as Code files to identify security issues and compliance violations in cloud infrastructure configurations.

Key Features of Snyk

Developer-Focused:

- Snyk integrates with developer workflows, making it easy for developers to identify and remediate security issues during the development process.

Dependency Analysis:

- Snyk analyzes dependencies in an application and provides insights into the security posture of third-party libraries and components.

Real-time Monitoring:

- Snyk offers continuous monitoring of applications and dependencies to alert organizations about newly discovered vulnerabilities and security risks.

Key Features of Snyk

Integration with CI/CD Pipelines:

- Snyk seamlessly integrates with continuous integration and continuous deployment (CI/CD) pipelines to automatically scan code and dependencies for vulnerabilities

Security for Multiple Languages:

- Snyk supports a wide range of programming languages, including JavaScript, Java, Python, Ruby, Go, and more.

Multi-Cloud and Multi-Platform Support:

- Snyk supports multiple cloud platforms, container orchestration tools, and version control systems, making it adaptable to various development and deployment environments.

How Synk Works?

- Snyk is a developer-friendly security platform designed to help organizations find and fix vulnerabilities and security issues in their software applications and open source dependencies.
- It integrates seamlessly into the software development lifecycle, making it easy for developers to identify and remediate security concerns.

How Snyk Works?

Scanning and Analysis

- Code Scanning
- Dependency Scanning
- Container Scanning
- IaC Scanning

Integration with development tools

Continuous Monitoring

Policy Enforcement

Benefits of Snyk?

- Early Detection of Vulnerabilities
- Developer-Friendly
- Comprehensive Dependency Scanning
- Container and IaC Security
- Custom Security Policies
- Integration with DevOps tools
- Collaboration and Communication
- Cost Reduction

Real World Use-Cases

- E-commerce Platform
- Healthcare Provider
- Financial Services
- Technology Startup
- Education Sector
- Adoption by DevOps team
- Open Source Project

Penetration Testing

- It is also referred as a “pen test” which is an authorised security test of an application/environment performed to evaluate its security
- Generally will actually exploit vulnerabilities that are found to see “how far” an exploit can go
- Generally a manual effort, can be difficult to automate
- Should be regularly scheduled, but if manual can be expensive so likely scheduled on roughly an annual basis
- Can be white-box, black-box or grey-box (a mix of white and black-box)

Vulnerability assessment/scanning

- Similar to a penetration test in that the objective is to identify vulnerabilities in an application/environment
- The main difference is that they do not actively exploit an identified vulnerabilities, rather they just report on the findings
- They can be automated as they do not need to be a manual process therefore are suited to CI/CD pipelines unlike a penetration test

CVEs

- CVE (**Common Vulnerabilities and Exposures**) identify, define and catalogue publicly disclosed cybersecurity vulnerabilities.
- It is important to stay aware of relevant CVEs for the tech stack that you or your company use.
- It is maintained by Mitre
- CVE-2021-44228 (CVE-Year-ID)
- Ratings: **Low, Medium, High** and **critical**

CVSS

- CVSS is **Common Vulnerability Scoring System**
- Standard for assessing severity of security vulnerability
- Scores calculated based on separate metrics, given a score 0-10 (one decimal place)
  - Low 0.1-3.9
  - Medium 4.0-6.9
  - High 7.0-8.9
  - Critical 9.0-10.0
- Used by CVEs
- https://www.first.org/cvss/calculator/3.0#CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

Key Security Principles

Defence in depth:

- Concept in which we never rely on one layer of security but rather we strive to have multiple layers of security.
- As an example, on a login page defence in depth might look like: **strong password requirements, CAPTCHA** shown when multiple incorrect attempts are made, **2FA**, then **email confirmation** requirements when logging in via an unknown IP/country. As you can see, even if the credentials are compromised it would not be sufficient to bypass the multiple layers of defence and gain access to the account

Key Security Principles

Least Privilege:

- Concept that everything within the environment must only have the minimum required access that is necessary in order to complete its purpose.
- For example, a service account running on a machine does not need to be a domain admin. Rather it could be restricted to not have an interactive login, restricted to no shell, restricted to specific file/folder permissions or only access e.g. in a database rather than full admin access

Key Security Principles

Authorisation vs authentication:

- It is important to distinguish the difference between authorization and authentication.
- **Authentication** is where you might be having the correct access to an application, but authorisation is what you are allowed to do once you are authenticated. Just because you are able to correctly authenticate does not mean you are allowed to do anything within the application.
- **Authorisation** restricts what you can do or can be used to help implement defence in depth and least privilege

CIA Triad

- **Confidentiality, integrity and availability**
- Simple, high level principles that form the foundation of any information security program

CIA Triad

- **Confidentiality:**

Ensuring data is kept safe secure and unavailable to unauthorised access. Multiple ways to protect confidentiality, an example of protecting confidentiality might be configuring authorisation for the initial access of the file then encrypting the file so even if it was made publically available you would need a key to decrypt and read the file

CIA Triad

Integrity:

Ensuring data is accurate and has not been tempered with, thus is trustworthy. To protect the integrity of data, we can leverage encryption, hashing, digital signatures and digital certificates. For example, we use digital certificates on websites to verify the authenticity of a website e.g. Google is actually Google

CIA Triad

Availability:

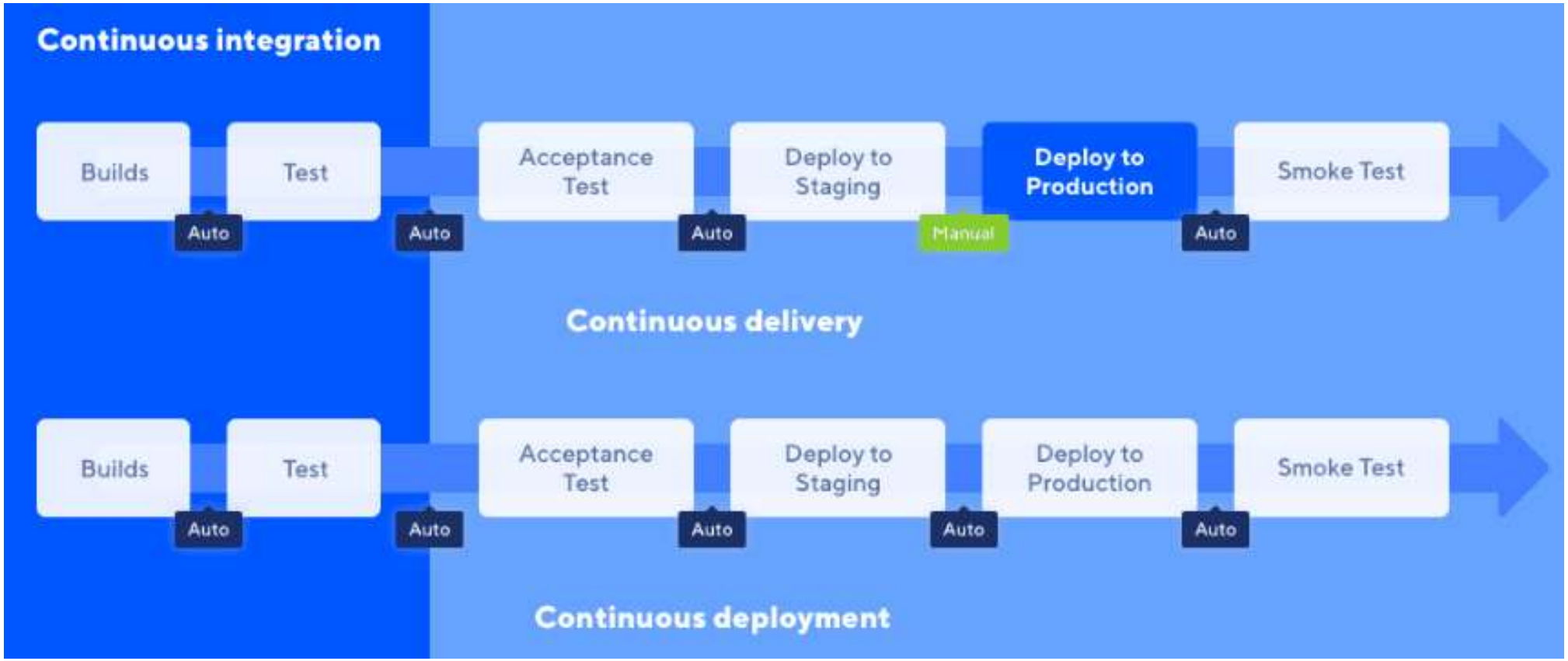
Ensuring that data and systems are available. For example, if an attacker employs a denial of service against a system this could potentially bring it offline thus impacting the availability if we do not sufficiently protect against such attacks

Module 8:
Security in Continuous Delivery (CD)

Continuous Delivery

- In continuous delivery (CD), all the code alterations are automatically implemented in either a testing or development environment right from the creation phase. This process is a straightforward prolongation of continuous integration.
- Simply put, it indicates that both your testing process and release method are automatic. Additionally, you only need a single click to apply the changes whenever you want.
- with continuous delivery, you can decide to release daily, weekly, fortnightly, or whatever suits your business requirements.

Continuous Delivery



Jenkins Recommendations

- Ensure it is not publicly available on the internet
- Ensure **authentication** is enabled
- Depending on environment, you may wish to plug into AD e.g SAML is supported or you may want to just use Jenkins local users (via Jenkins own database) if there is only a handful of users
- Ensure plugins are up to date
- Ensure Jenkins version is up to date (include Jenkins in automatic updates)
- Ensure Java version that Jenkins is using is up to date
- Ensure Jenkins service runs as dedicated least privilege user

Jenkins Recommendations

- Ensure **CSRF protection** is enabled.
- **Cross-Site Request Forgery (CSRF)** protection is crucial for securing web applications, including Jenkins. CSRF protection helps prevent attackers from executing unauthorized actions on behalf of authenticated users.
- CSRF attacks typically involve an attacker crafting a malicious request and tricking the victim into submitting it without their knowledge or consent.
- Manage Jenkins → Configure Global security

Jenkins Recommendations

- Disable Jenkins SSHD server (ensures that the underlying servers SSH is hardened as per previous SSH section as well as general server hardening applied, check CIS benchmarks for specific OS)
- Manage Jenkins → Configure Global Security → SSH Server → SSHD Port → Disable

Jenkins Recommendations

- In Jenkins, controlling access between **agents** and the **controller** is crucial for maintaining a secure and well-managed CI/CD environment.
- Properly configuring access control ensures that only authorized agents can connect to the Jenkins controller, adding an extra layer of security to your CI/CD environment.
- One option for securing the connection is to use a random port for each agent. This means that agents dynamically choose a port to connect to the controller.
- Enable agent → controller access control
- Manage Jenkins → Configure Global Security

Jenkins Recommendations

- Ensure access control is configured (authorisation)
- Ensuring access control through proper authorization mechanisms is crucial for securing Jenkins. Authorization controls dictate who has permission to perform specific actions within Jenkins. Jenkins provides various authorization strategies, and you can configure them based on your organization's security requirements.
- Select an appropriate authorization strategy based on your organization's needs. Common strategies include:
  - **Legacy Authorization**: Simplest strategy with a single global configuration.
  - **Matrix-based Security**: Provides a matrix where you can specify permissions for each user or group.
  - **Project-based Matrix Authorization Strategy**: Similar to Matrix-based Security but allows permissions to be defined per project.
  - **Role-based Authorization Strategy**: Allows more sophisticated role-based access control.
- Manage Jenkins → Configure Global Security

Jenkins Recommendations

- Ensure the TLS config is secure
- TLS 1.3 only
  - TLS_AES_128_GCM_SHA256
  - TLS_AES_256_GCM_SHA384
  - TLS_CHACHA20_POLY1305_SHA256
- Strong cipher suites are forward secret only
- Strong certificate e.g. ECDSA or RSA 2048+
- HSTS Max-age = 2 years

Jenkins Recommendations

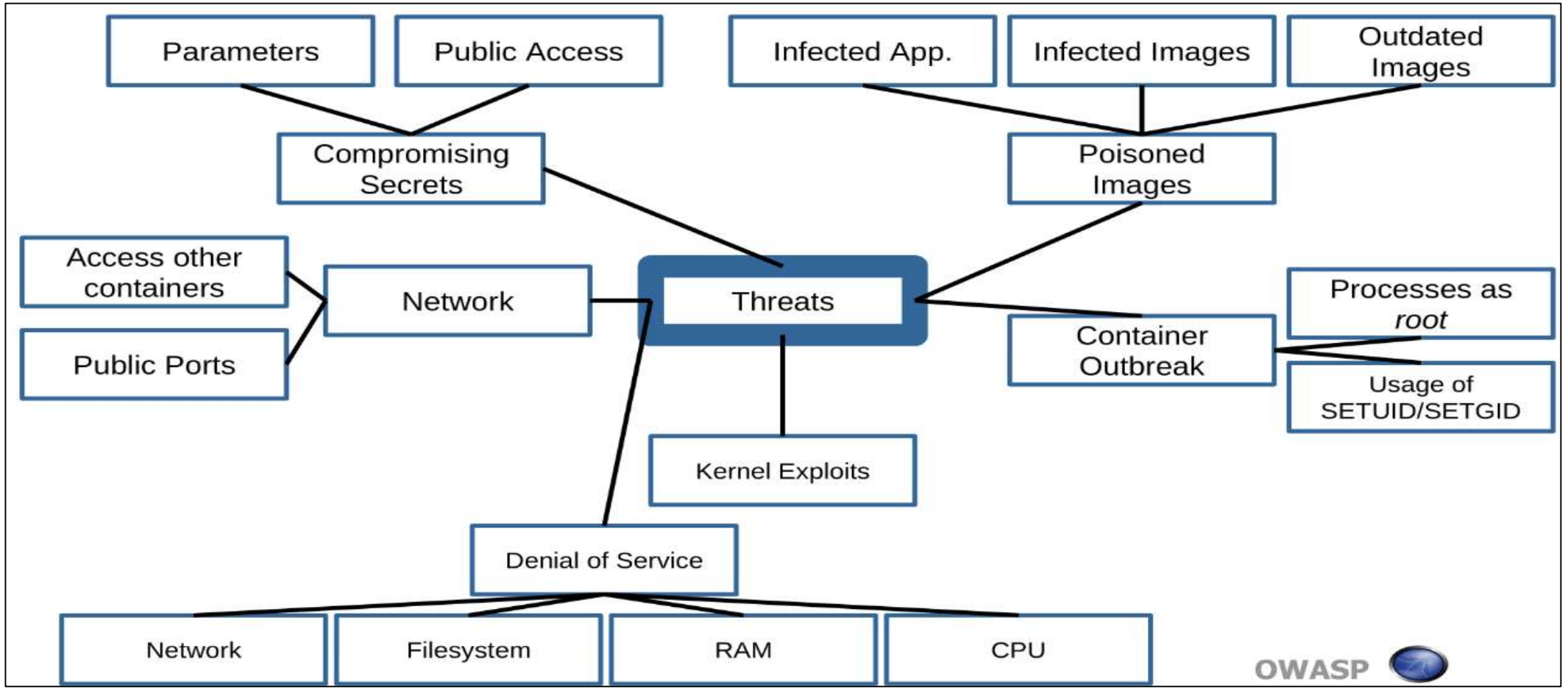
Ensuring proper HTTP headers are set is an important aspect of web security and can help protect your Jenkins instance from various vulnerabilities and attacks. Below are some common HTTP headers and recommendations for setting them in Jenkins:

- **Strict-Transport-Security (HSTS):**
  - HSTS header instructs browsers to only connect to Jenkins over HTTPS, reducing the risk of man-in-the-middle attacks.
  - To set HSTS in Jenkins, you can use a reverse proxy or configure it directly in Jenkins, depending on your setup.
- **CSP header helps prevent various types of attacks, including Cross-Site Scripting (XSS).**
- **The X-Content-Type-Options header prevents browsers from interpreting files as a different MIME type.**
- **The X-Frame-Options header prevents the Jenkins pages from being embedded within an iframe, reducing the risk of clickjacking attacks.**
- **The Referrer-Policy header controls how much referrer information is sent with requests.**

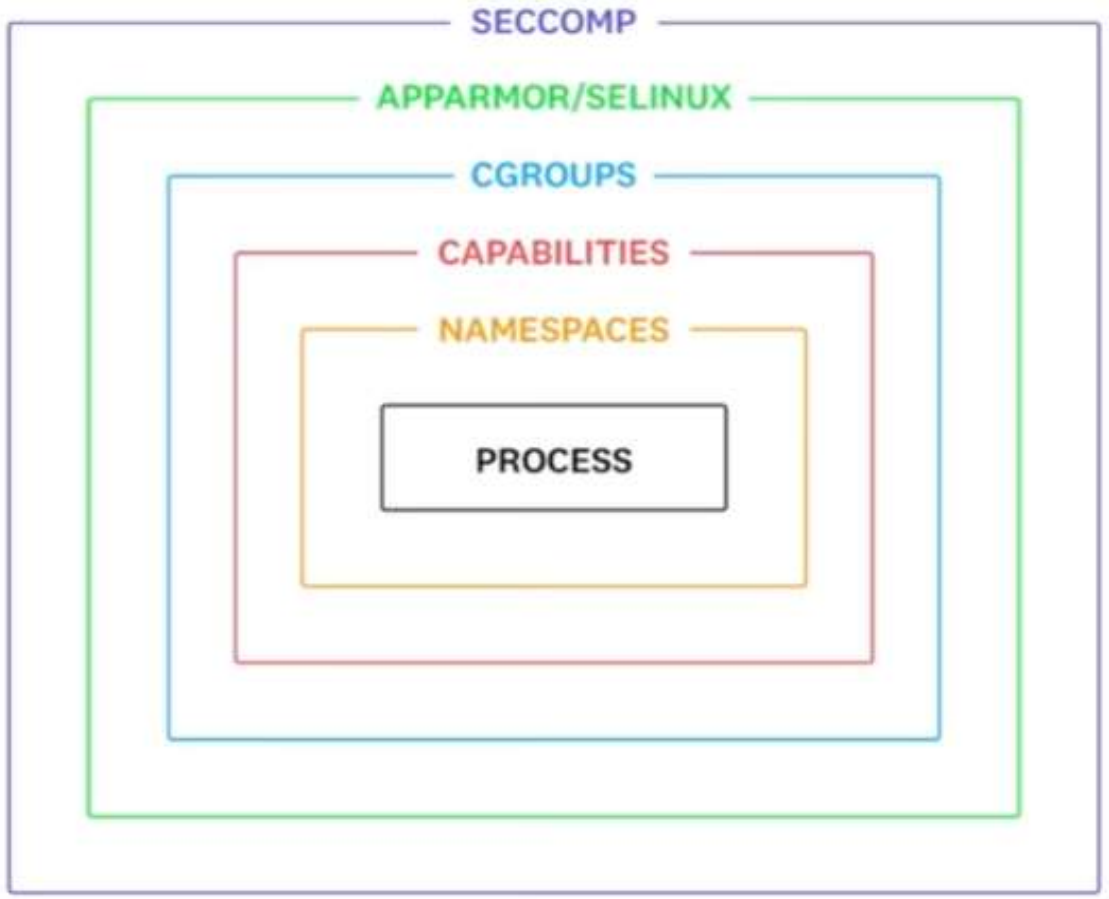
Reference OWASP Secure Header for further details and check periodically as it keeps changing over time. (https://owasp.org/www-project-secure-headers/)

Module 9:
Container Security

Threats in Docker



Containers are just processes



Containers are just processes

Create a container

docker run –dt –name web nginx

```
root@ip-172-31-11-94:~# ps -fC nginx
UID          PID    PPID  C STIME TTY          TIME CMD
root        6115    6093  0 19:39 pts/0    00:00:00 nginx: master process nginx -g daemon off;
systemd+    6161    6115  0 19:39 pts/0    00:00:00 nginx: worker process
systemd+    6162    6115  0 19:39 pts/0    00:00:00 nginx: worker process
```

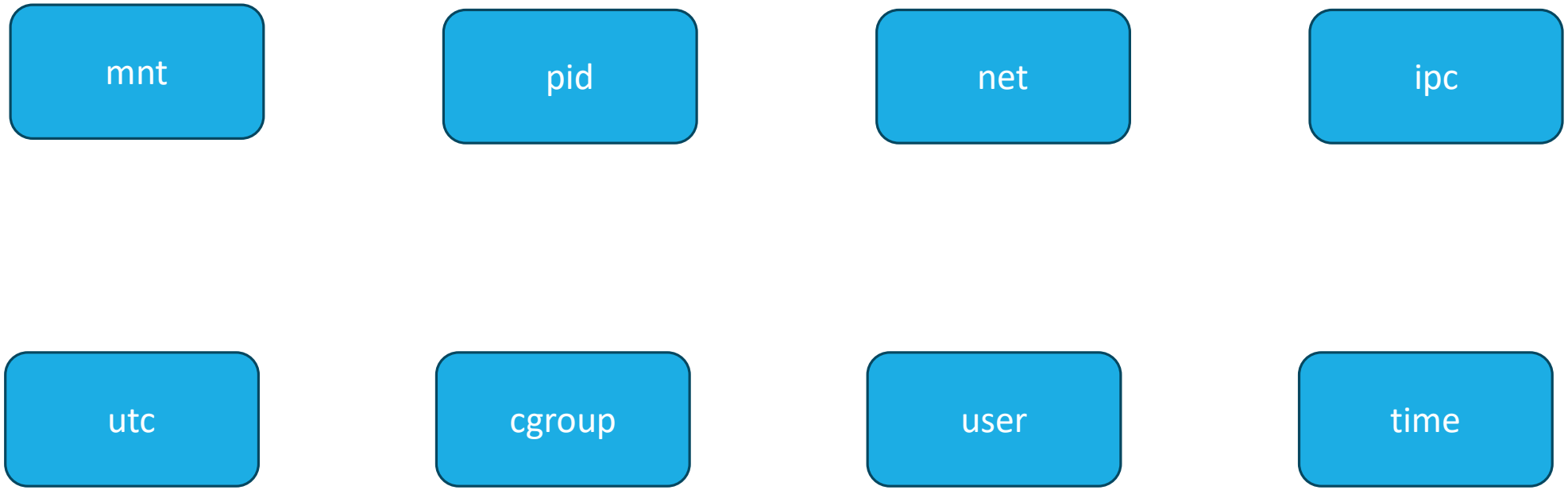
docker exec web touch Sandeep

ls /proc/6115/root

```
root@ip-172-31-11-94:~# ls /proc/6115/root
bin   dev                 docker-entrypoint.sh  home  lib32  libx32  mnt  proc  run      sbin  sys  usr
boot  docker-entrypoint.d  etc                   lib   lib64  media   opt  root  sandeep  srv   tmp  var
root@ip-172-31-11-94:~#
```

Container Namespaces

Each namespace provides a separate and isolated view of a particular system resource. 6 namespaces are enabled by default out of 8. User is not enabled by default and time is not there is containers as yet



Container Namespaces

```
root@ip-172-31-11-94:~# lsns
        NS TYPE   NPROCS   PID USER            COMMAND
4026531834 time      112     1 root            /sbin/init
4026531835 cgroup    109     1 root            /sbin/init
4026531836 pid       109     1 root            /sbin/init
4026531837 user      112     1 root            /sbin/init
4026531838 uts       105     1 root            /sbin/init
4026531839 ipc       109     1 root            /sbin/init
4026531840 net       109     1 root            /sbin/init
4026531841 mnt       101     1 root            /sbin/init
4026531862 mnt         1    25 root            kdevtmpfs
4026532271 mnt         1   174 root            /lib/systemd/systemd-udevd
4026532272 uts         1   174 root            /lib/systemd/systemd-udevd
4026532274 mnt         1   340 systemd-network /lib/systemd/systemd-networkd
4026532275 mnt         1   342 systemd-resolve /lib/systemd/systemd-resolved
4026532276 mnt         2   455 _chrony         /usr/sbin/chronyd -F 1
4026532277 uts         2   455 _chrony         /usr/sbin/chronyd -F 1
4026532283 mnt         3  6115 root            nginx: master process nginx -g daemon off;
4026532284 uts         3  6115 root            nginx: master process nginx -g daemon off;
4026532285 ipc         3  6115 root            nginx: master process nginx -g daemon off;
4026532286 pid         3  6115 root            nginx: master process nginx -g daemon off;
4026532287 net         3  6115 root            nginx: master process nginx -g daemon off;
4026532334 mnt         1   465 root            /usr/sbin/irqbalance --foreground
4026532335 mnt         1   482 root            /lib/systemd/systemd-logind
4026532336 uts         1   482 root            /lib/systemd/systemd-logind
4026532346 cgroup      3  6115 root            nginx: master process nginx -g daemon off;
```

Container Namespaces

Check the mnt namespace using the pid of the container

findmnt –N 6115 or cat /proc/6115/mountinfo

```
root@ip-172-31-11-94:~# findmnt -N 6115
TARGET                  SOURCE                 FSTYPE  OPTIONS
/                       overlay                overlay rw,relatime,lowerdir=/var/lib/docker/0.0/overlay2/l/5KW3WQXDODYFAREB2IWNJENDRL:/var/lib/docker/0.0/overl
├─/proc                 proc                   proc    rw,nosuid,nodev,noexec,relatime
│ ├─/proc/bus           proc[/bus]             proc    ro,nosuid,nodev,noexec,relatime
│ ├─/proc/fs            proc[/fs]              proc    ro,nosuid,nodev,noexec,relatime
│ ├─/proc/irq           proc[/irq]             proc    ro,nosuid,nodev,noexec,relatime
│ ├─/proc/sys           proc[/sys]             proc    ro,nosuid,nodev,noexec,relatime
│ ├─/proc/sysrq-trigger proc[/sysrq-trigger]   proc    ro,nosuid,nodev,noexec,relatime
│ ├─/proc/acpi          tmpfs                  tmpfs   ro,relatime,inode64
│ ├─/proc/kcore         tmpfs[/null]           tmpfs   rw,nosuid,size=65536k,mode=755,inode64
│ ├─/proc/keys          tmpfs[/null]           tmpfs   rw,nosuid,size=65536k,mode=755,inode64
│ ├─/proc/timer_list    tmpfs[/null]           tmpfs   rw,nosuid,size=65536k,mode=755,inode64
│ └─/proc/scsi          tmpfs                  tmpfs   ro,relatime,inode64
├─/dev                  tmpfs                  tmpfs   rw,nosuid,size=65536k,mode=755,inode64
│ ├─/dev/console        devpts[/0]             devpts  rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=666
│ ├─/dev/pts            devpts                 devpts  rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=666
│ ├─/dev/mqueue         mqueue                 mqueue  rw,nosuid,nodev,noexec,relatime
│ └─/dev/shm            shm                    tmpfs   rw,nosuid,nodev,noexec,relatime,size=65536k,inode64
├─/sys                  sysfs                  sysfs   ro,nosuid,nodev,noexec,relatime
│ ├─/sys/firmware       tmpfs                  tmpfs   ro,relatime,inode64
│ └─/sys/fs/cgroup      cgroup[/system.slice/docker-9a044983ff1d996827222a79eb3f4a2ebe6e261349f339535f232d22b6a6467d.scope]
│                                              cgroup2 ro,nosuid,nodev,noexec,relatime,nsdelegate,memory_recursiveprot
├─/etc/resolv.conf      /dev/xvda1[/var/lib/docker/0.0/containers/9a044983ff1d996827222a79eb3f4a2ebe6e261349f339535f232d22b6a6467d/resolv.conf]
│                                              ext4    rw,relatime,discard,errors=remount-ro
├─/etc/hostname         /dev/xvda1[/var/lib/docker/0.0/containers/9a044983ff1d996827222a79eb3f4a2ebe6e261349f339535f232d22b6a6467d/hostname]
│                                              ext4    rw,relatime,discard,errors=remount-ro
└─/etc/hosts            /dev/xvda1[/var/lib/docker/0.0/containers/9a044983ff1d996827222a79eb3f4a2ebe6e261349f339535f232d22b6a6467d/hosts]
                                               ext4    rw,relatime,discard,errors=remount-ro
```

Container Namespaces

We can also use nsenter in order to interact with the namespace.

nsenter --target 6115 --mount ls /

```
root@ip-172-31-11-94:~# nsenter --target 6115 --mount ls /
bin   dev                 docker-entrypoint.sh  home  lib32  libx32  mnt  proc  run      sbin  sys  usr
boot  docker-entrypoint.d  etc                   lib   lib64  media   opt  root  sandeep  srv   tmp  var
root@ip-172-31-11-94:~#
```

The PID namespace

The PID (Process ID) namespace in Linux is a feature that allows processes to have their own isolated view of the process hierarchy.

docker run --name busybox -d busybox top

nsenter --target 6308 -m -p ps -ef (6308 is the pid of the above container)

This allows you to see the list of processes within the context of the specified namespaces.

```
root@ip-172-31-11-94:~# nsenter --target 6308 -m -p ps -ef
PID   USER     TIME   COMMAND
    1 root       0:00 top
    8 root       0:00 ps -ef
root@ip-172-31-11-94:~#
```

The PID namespace

The unshare command in Linux is used to create a new namespace and run a command in that namespace.

unshare --pid --fork --mount-proc /bin/bash

```
root@ip-172-31-11-94:~# unshare --pid --fork --mount-proc /bin/bash
root@ip-172-31-11-94:~#
root@ip-172-31-11-94:~# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 20:32 pts/1    00:00:00 /bin/bash
root           8       1  0 20:32 pts/1    00:00:00 ps -ef
root@ip-172-31-11-94:~#
```

The PID namespace

We can also join the pid namespace of the container with another container

Docker run –dt –name web nginx

docker run -it --pid=container:web ubuntu

```
root@ip-172-31-11-94:~#
root@ip-172-31-11-94:~# docker run -it --pid=container:web ubuntu
root@e58feaf00f5e:/#
root@e58feaf00f5e:/# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 19:39 pts/0    00:00:00 nginx: master process nginx -g daemon off;
101           30       1  0 19:39 pts/0    00:00:00 nginx: worker process
101           31       1  0 19:39 pts/0    00:00:00 nginx: worker process
root          39       0  0 20:38 pts/0    00:00:00 /bin/bash
root          47      39  0 20:38 pts/0    00:00:00 ps -ef
root@e58feaf00f5e:/#
```

Network Namespace

nsenter --target 6115 -n ip addr

```
root@ip-172-31-11-94:~#
root@ip-172-31-11-94:~# nsenter --target 6115 -n ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
36: eth0@if37: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
root@ip-172-31-11-94:~#
```

Network Namespace

docker run -it --network=container:web ubuntu

```
root@9a044983ff1d:/# netstat -tunap
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      -
tcp        0      0 172.17.0.2:34786        185.125.190.39:80       TIME_WAIT   -
tcp        0      0 172.17.0.2:47660        91.189.91.81:80         TIME_WAIT   -
tcp        0      0 172.17.0.2:33234        18.225.36.18:80         TIME_WAIT   -
tcp        0      0 172.17.0.2:48410        91.189.91.82:80         TIME_WAIT   -
tcp6       0      0 :::80                   :::*                    LISTEN      -
root@9a044983ff1d:/# root@ip-172-31-11-94:~#
```

1. Secure User Mapping

- Most often the application within the container runs with the default administrative privileges: **root**.
- This violates the least privilege principle and gives an attacker better chances further extending his activities if he manages to break out of the application into the container.
- From the host perspective the application should never run as root.
- By default all the containers will run as a root

1. Secure User Mapping

However containers still have some default protection like Linux capabilities, AppArmor or SELinux, but running a container as a root user removes one layer of protection.

Never use –privileged flag, it gives so-called capabilities (D04) to the container and it can access host devices, including disks and also has access to the /sys and /proc filesystem.

And the container can even load kernel modules on the host.

By default containers are unprivileged so you would have to configure them explicitly to run privileged.

1. Secure User Mapping

Method 1: Creating a user and a group

Add below to the Dockefile. Sets the home directory for the user to /bin/false. This effectively disables the user's ability to log in.

RUN groupadd -r dummy && useradd -r -d /bin/false -g dummy dummy

WORKDIR /usr/src/app

RUN chown -R dummy:dummy /usr/src/app

USER dummy

https://github.com/CloudSihmar/spring-boot-hello-world.git

1. Secure User Mapping

Method 2: Using Linux Namespaces

There are 8 namespaces available in Linux and six are enabled by default. Each of those used to isolate different resource.

- Mnt
- Pid
- Net
- Ipc
- Uts
- Cgroup
- User
- time

1. Secure User Mapping

Method 2: Using Linux Namespaces

User namespace is not enabled by default, it is useful because it helps you decouple the user ID inside the container from user ID on the host machine. It allows you to appear to be the root inside a container without giving you the privileges of the root user on the underlying host. It is a big win for security as you are not running as root.

Run the daemon with the file /etc/docker/daemon.json

```
{
  "userns-remap": "testuser"
}
```

2. Linux Capabilities

Capabilities are a way to assign specific privileges to a running process. They allow us to have more fine-grained control over the privileges that processes have on a Linux system.

In Unix, we have two main controls: superuser (root) and normal user (non-root).

The UID stands for user identifier and is used to determine what a user can do within the system.

The UID of the root user is set to 0 meaning it can do (almost) anything and has the maximum number of privileges.

A non-zero UID signifies that it’s a normal user who generally does not have permissions to install software, modify system files, and more.

2. Linux Capabilities

The Linux kernel is able to break down the privileges of the root user into distinct units referred to as capabilities.

For example:

CAP_CHOWN capability is what allows the root use to make arbitrary changes to file UIDs and GIDs.

The CAP_DAC_OVERRIDE capability allows the root user to bypass kernel permission checks on file read, write and execute operations.

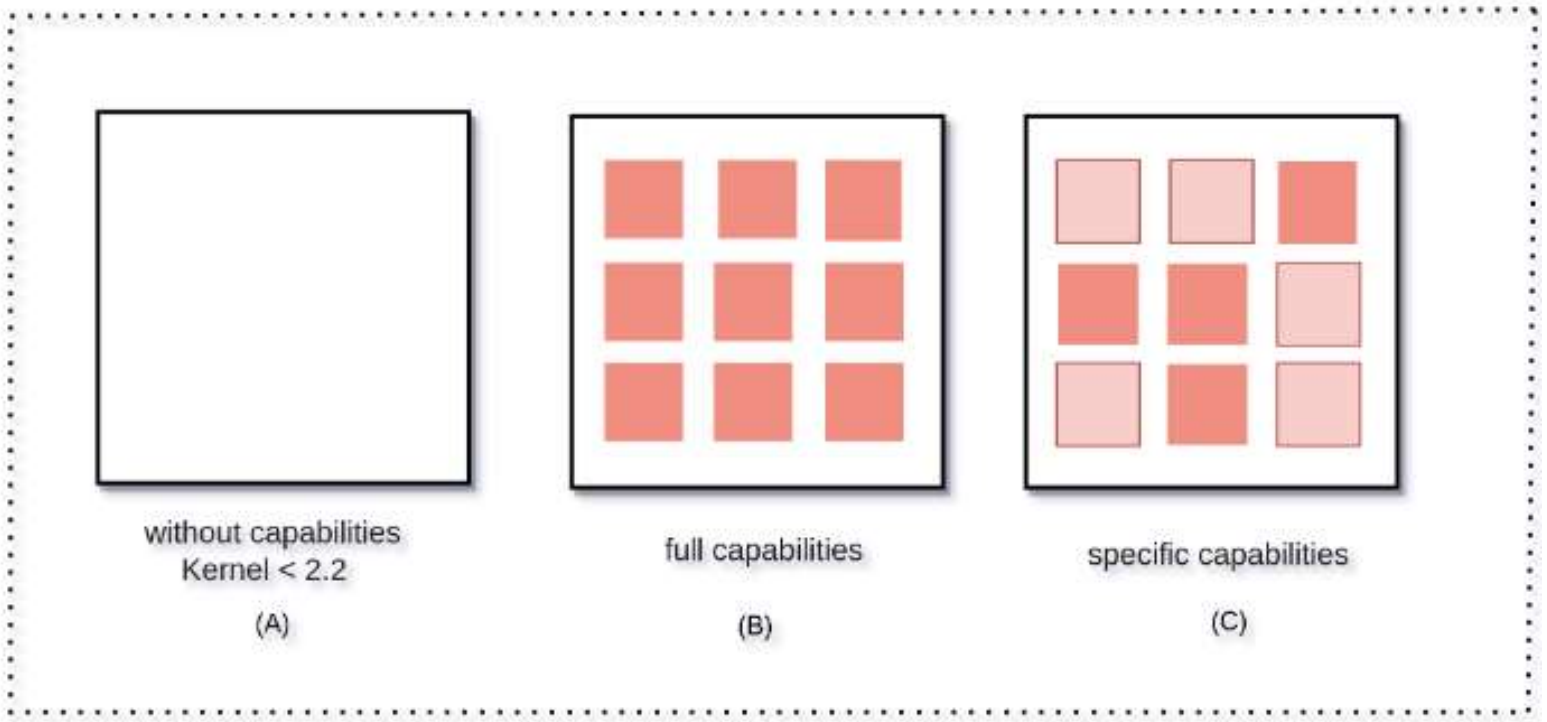
Almost all of the special powers associated with the Linux root user are broken down into individual capabilities.

This breaking down of root privileges into granular capabilities allows you to:

- Remove individual capabilities from the root user account, making it less powerful/dangerous.
- Add privileges to non-root users at a very granular level.

2. Linux Capabilities

You need capabilities to reduce the attack surface. You can restrict permissions using Linux capabilities without giving complete permissions.



2. Linux Capabilities

Lets try to change the date inside a container.

docker run -it ubuntu /bin/bash

date -s "2024-02-10 12:34:56“

```
root@98d3a66413ba:/#  date -s "2024-02-10 12:34:56"
date: cannot set date: Operation not permitted
Sat Feb 10 12:34:56 UTC 2024
root@98d3a66413ba:/#
```

Time namespace is not used in the container, so it uses the same time as the host machine.

The container does not have the capability to change the time

2. Linux Capabilities

pscap:

This shows as any process on the host which either has all the capabilities or subset of capabilities

apt install libcap-ng-utils

2. Linux Capabilities

filecap utility:

It will scan all the files which has the capabilities assigned to them.

```
root@ip-172-31-86-135:~#
root@ip-172-31-86-135:~# filecap -a 2>/dev/null
file                capabilities
/snap/core20/2015/usr/bin/ping     net_raw
/usr/lib/x86_64-linux-gnu/gstreamer1.0/gstreamer-1.0/gst-ptp-helper     net_bind_service, net_admin
/usr/bin/ping     net_raw
/usr/bin/mtr-packet     net_raw
root@ip-172-31-86-135:~#
```

2. Docker Capabilities

- Docker imposes certain limitations that make working with capabilities much simpler.
- Docker sets the bounding set before starting a container. You can use Docker commands to add or remove capabilities to or from the bounding set.
- By default, Docker drops all capabilities except those needed, using a whitelist approach.
- Docker provides the default capabilities allow the container workflow run without any problem without allowing for the privileged escalation.

2. Docker Capabilities

docker run –dt nginx

pscap | grep –i nginx

```
root@ip-172-31-86-135:~# pscap | grep -i nginx
3158  3179  root        nginx             chown, dac_override, fowner, fsetid, kill, setgid, setuid, setpcap, net_bind_service,
net_raw, sys_chroot, mknod, audit_write, setfcap
root@ip-172-31-86-135:~#
root@ip-172-31-86-135:~#
```

2. Docker Capabilities

	Capabilities	Definition
1	CHOWN	Make arbitrary changes to file UIDs and GIDs
2	DAC_OVERRIDE	Discretionary access control (DAC) - Bypass file read, write, and execute permission checks
3	FSETID	Don’t clear set-user-ID and set-group-ID mode bits when a file is modified; set the set-group-ID bit for a file whose GID does not match the file system or any of the supplementary GIDs of the calling process.
4	FOWNER	Bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file, excluding those operations covered by CAP_DAC_OVERRIDE and CAP_DAC_READ_SEARCH.
5	MKNOD	MKNOD - Create special files using mknod(2)
6	NET_RAW	Use RAW and PACKET sockets; bind to any address for transparent proxying.
7	SETGID	Make arbitrary manipulations of process GIDs and supplementary GID list; forge GID when passing socket credentials via UNIX domain sockets; write a group ID mapping in a user namespace.

2. Docker Capabilities

	Capabilities	Definition
8	SETUID	Make arbitrary manipulations of process UIDs; forge UID when passing socket credentials via UNIX domain sockets; write a user ID mapping in a user namespace.
9	SETFCAP	Set file capabilities.
10	SETPCAP	If file capabilities are not supported: grant or remove any capability in the caller's permitted capability set to or from any other process.
11	NET_BIND_SERVICE	Bind a socket to Internet domain privileged ports (port numbers less than 1024).
12	SYS_CHROOT	Use chroot(2) to change to a different root directory.
13	KILL	Bypass permission checks for sending signals. This includes use of the ioctl(2) KDSIGACCEPT operation.
14	AUDIT_WRITE	Write records to kernel auditing log.

2. Docker Capabilities

Lets check inside a container what capabilities it has been assigned using the amicontained utility.

docker run -it cloudsihmar/container-tools:latest /bin/bash

```
root@ip-172-31-86-135:~# docker run -it  cloudsihmar/container-tools:latest /bin/bash
root in container ID 4c664a95ee1e in /~
# amicontained
Container Runtime: not-found
Has Namespaces:
        pid: true
        user: false
AppArmor Profile: docker-default (enforce)
Capabilities:
        BOUNDING -> chown dac_override fowner fsetid kill setgid setuid setpcap net_bind_service net_raw sys_chroot mknod audit_write setfcap
Seccomp: filtering
Blocked Syscalls (54):
        MSGRCV SYSLOG SETSID USELIB USTAT SYSFS VHANGUP PIVOT_ROOT _SYSCTL ACCT SETTIMEOFDAY MOUNT UMOUNT2 SWAPON SWAPOFF REBOOT SETHOSTNAME SE
TDOMAINNAME IOPL IOPERM CREATE_MODULE INIT_MODULE DELETE_MODULE GET_KERNEL_SYMS QUERY_MODULE QUOTACTL NFSSERVCTL GETPMSG PUTPMSG AFS_SYSCALL TU
XCALL SECURITY LOOKUP_DCOOKIE CLOCK_SETTIME VSERVER MBIND SET_MEMPOLICY GET_MEMPOLICY KEXEC_LOAD ADD_KEY REQUEST_KEY KEYCTL MIGRATE_PAGES UNSHA
RE MOVE_PAGES PERF_EVENT_OPEN FANOTIFY_INIT OPEN_BY_HANDLE_AT SETNS KCMP FINIT_MODULE KEXEC_FILE_LOAD BPF USERFAULTFD
Looking for Docker.sock
root in container ID 4c664a95ee1e in /~
#
```

2. Docker Capabilities

you have 3 high level options for using capabilities:

- Run containers as root with a large set of capabilities and try to manage capabilities within your container manually.
- Run containers as root with limited capabilities and never change them within a container.
- Run containers as an unprivileged user with no capabilities.

Docker Capabilities

- Docker containers are provided with a set of capabilities by default
- Depending on your application, you may be able to drop some or all of these capabilities to help harden your containers.
- It's worth remembering that capabilities are rights granted to the root user.
- This means that if your application runs perfectly well as a non-root user, it should run fine in a container without any capabilities. In those cases, you can just drop all capabilities and it should work fine.
- It's also worth noting that there are a couple of common cases when you traditionally would have needed capabilities, but no longer need them anymore.

2. Docker Capabilities : net_raw

Granting the CAP_NET_RAW capability to a process can introduce security concerns because it allows the process to create raw sockets, which enables low-level manipulation of network packets.

Raw sockets provide direct access to network protocols, and mishandling or misuse of this capability can lead to security vulnerabilities. Containers use this for the ping purpose but Linux exposed a setting specially for ICMP so container do not need this capability.

docker run -it --cap-drop=net_raw cloudsihmar/container-tools:latest /bin/bash

```
root@ip-172-31-86-135:~# docker run -it  --cap-drop=net_raw cloudsihmar/container-tools:latest /bin/bash
root in container ID dbaf363799d7 in /~
# amicontained
Container Runtime: not-found
Has Namespaces:
        pid: true
        user: false
AppArmor Profile: docker-default (enforce)
Capabilities:
        BOUNDING -> chown dac_override fowner fsetid kill setgid setuid setpcap net_bind_service sys_chroot mknod audit_write setfcap
Seccomp: filtering
Blocked Syscalls (54):
        MSGRCV SYSLOG SETSID USELIB USTAT SYSFS VHANGUP PIVOT_ROOT _SYSCTL ACCT SETTIMEOFDAY MOUNT UMOUNT2 SWAPON SWAPOFF REBOOT SETHOSTNAME SE
TDOMAINNAME IOPL IOPERM CREATE_MODULE INIT_MODULE DELETE_MODULE GET_KERNEL_SYMS QUERY_MODULE QUOTACTL NFSSERVCTL GETPMSG PUTPMSG AFS_SYSCALL TU
XCALL SECURITY LOOKUP_DCOOKIE CLOCK_SETTIME VSERVER MBIND SET_MEMPOLICY GET_MEMPOLICY KEXEC_LOAD ADD_KEY REQUEST_KEY KEYCTL MIGRATE_PAGES UNSHA
RE MOVE_PAGES PERF_EVENT_OPEN FANOTIFY_INIT OPEN_BY_HANDLE_AT SETNS KCMP FINIT_MODULE KEXEC_FILE_LOAD BPF USERFAULTFD
Looking for Docker.sock
root in container ID dbaf363799d7 in /~
# sysctl net.ipv4.ping_group_range
net.ipv4.ping_group_range = 0   2147483647
root in container ID dbaf363799d7 in /~
```

2. Docker Capabilities : net_bind_service

The CAP_NET_BIND_SERVICE capability in Linux allows a process to bind to privileged ports (ports below 1024) without requiring root privileges. While this capability is sometimes necessary for certain applications, it introduces potential security risks that need to be carefully considered.

A process with CAP_NET_BIND_SERVICE could potentially hijack a service by binding to its port.

docker run -it --cap-drop=net_bind_service cloudsihmar/container-tools:latest /bin/bash

```
root@ip-172-31-86-135:~# docker run -it  --cap-drop=net_bind_service cloudsihmar/container-tools:latest /bin/bash
root in container ID b205117eb456 in /~
# amicontained
Container Runtime: not-found
Has Namespaces:
        pid: true
        user: false
AppArmor Profile: docker-default (enforce)
Capabilities:
        BOUNDING -> chown dac_override fowner fsetid kill setgid setuid setpcap net_raw sys_chroot mknod audit_write setfcap
Seccomp: filtering
Blocked Syscalls (54):
        MSGRCV SYSLOG SETSID USELIB USTAT SYSFS VHANGUP PIVOT_ROOT _SYSCTL ACCT SETTIMEOFDAY MOUNT UMOUNT2 SWAPON SWAPOFF REBOOT SETHOSTNAME SE
TDOMAINNAME IOPL IOPERM CREATE_MODULE INIT_MODULE DELETE_MODULE GET_KERNEL_SYMS QUERY_MODULE QUOTACTL NFSSERVCTL GETPMSG PUTPMSG AFS_SYSCALL TU
XCALL SECURITY LOOKUP_DCOOKIE CLOCK_SETTIME VSERVER MBIND SET_MEMPOLICY GET_MEMPOLICY KEXEC_LOAD ADD_KEY REQUEST_KEY KEYCTL MIGRATE_PAGES UNSHA
RE MOVE_PAGES PERF_EVENT_OPEN FANOTIFY_INIT OPEN_BY_HANDLE_AT SETNS KCMP FINIT_MODULE KEXEC_FILE_LOAD BPF USERFAULTFD
Looking for Docker.sock
root in container ID b205117eb456 in /~
# sysctl net.ipv4.ip_unprivileged_port_start
net.ipv4.ip_unprivileged_port_start = 0
root in container ID b205117eb456 in /~
#
```

2. Docker Capabilities : Drop all

Run the container with no capabilities:

docker run -it --cap-drop=all cloudsihmar/container-tools:latest /bin/bash

```
root@ip-172-31-86-135:~# docker run -it  --cap-drop=all cloudsihmar/container-tools:latest /bin/bash
root in container ID 61ec113fe66b in /~
# amicontained
Container Runtime: not-found
Has Namespaces:
        pid: true
        user: false
AppArmor Profile: docker-default (enforce)
Capabilities:
Seccomp: filtering
Blocked Syscalls (56):
        MSGRCV SYSLOG SETSID SETGROUPS USELIB USTAT SYSFS VHANGUP PIVOT_ROOT _SYSCTL CHROOT ACCT SETTIMEOFDAY MOUNT UMOUNT2 SWAPON SWAPOFF REBO
OT SETHOSTNAME SETDOMAINNAME IOPL IOPERM CREATE_MODULE INIT_MODULE DELETE_MODULE GET_KERNEL_SYMS QUERY_MODULE QUOTACTL NFSSERVCTL GETPMSG PUTPM
SG AFS_SYSCALL TUXCALL SECURITY LOOKUP_DCOOKIE CLOCK_SETTIME VSERVER MBIND SET_MEMPOLICY GET_MEMPOLICY KEXEC_LOAD ADD_KEY REQUEST_KEY KEYCTL MI
GRATE_PAGES UNSHARE MOVE_PAGES PERF_EVENT_OPEN FANOTIFY_INIT OPEN_BY_HANDLE_AT SETNS KCMP FINIT_MODULE KEXEC_FILE_LOAD BPF USERFAULTFD
Looking for Docker.sock
root in container ID 61ec113fe66b in /~
```

2. Docker Capabilities

Check the ownership capabilities if applied or not

```
root@ip-172-31-86-135:~# docker exec -it 7bfbb9d598ea /bin/bash
root in container ID 7bfbb9d598ea in /~
# chown nobody /
chown: changing ownership of '/': Operation not permitted
root in container ID 7bfbb9d598ea in /~
```

2. Docker Capabilities

docker container run --rm -it --cap-drop ALL --cap-add CHOWN alpine chown nobody /

The operation succeeds because although you dropped all capabilities for the container's root account, you added the chown capability back. The chown capability is all that is needed to change the ownership of a file.

docker container run --rm -it --cap-add chown -u nobody alpine chown nobody /

The above command fails because Docker does not yet support adding capabilities to non-root users.

3. Docker cGroups

- A single misbehaving program could consume all available resources, causing the entire system to crash.
- To tackle this problem, Linux relies on control groups (cgroups) to manage each process's access to resources, such as CPU and memory.
- Docker and other containerization tools use cgroups to restrict the resources that containers can use, which can help avoid "noisy neighbor" issues.
- cGroups can help alleviate denial-of-service risks.

3. Docker cGroups

Limit the CPU resource of the container

docker run --name stress --cpu 0.5 cloudsihmar/stress:latest -c 2

```
top - 07:31:25 up  2:09,  2 users,  load average: 0.06, 0.05, 0.01
Tasks: 123 total,   3 running, 120 sleeping,   0 stopped,   0 zombie
%Cpu(s): 25.1 us,  0.0 sy,  0.0 ni, 74.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3904.5 total,    231.6 free,    315.1 used,   3357.8 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.   3293.0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  15668 root      20   0    3704    256    256 R  25.9   0.0   0:17.97 stress
  15667 root      20   0    3704    256    256 R  23.6   0.0   0:17.00 stress
      1 root      20   0   18816  11392   8192 S   0.0   0.3   0:07.42 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
```

3. Docker cGroups

Using cgroups to defeat fork bombs:

- A common denial-of-service attack on Linux systems is known as a fork bomb, which occurs when an attacker generates a very large number of processes, ultimately depleting the system's resources.
- By default, containers (and other Linux processes) are not restricted in terms of how many new processes they can generate, which means that any process can create a fork bomb.
- Cgroups have the ability to restrict the number of processes that can be spawned, which effectively safeguards the host from a fork bomb attack.

3. Docker cGroups

Set the pid limits:

docker run -it --pids-limit 10 ubuntu /bin/bash

:(){ :|: & };:

The container reaches the limit of 10 processes, and errors are displayed. However, the underlying host will remain responsive, preventing the denial-of-service attack.

```
root@ip-172-31-86-135:~# docker run -it --pids-limit 10 ubuntu /bin/bash
root@d0d32e0cd80e:/# :(){ :|: & };:
[1] 10
root@d0d32e0cd80e:/# bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
```

4. Mandatory Access Control Systems

There are two access controls available:

MAC (Mandatory Access Control) :

- access control is centrally administered and mandated by a security policy defined by system administrators or security policies.
- MAC enforces the principle of least privilege, meaning that users and processes are granted only the minimum access rights necessary to perform their tasks.
- Access is determined by system-wide security policies, and users cannot override them.
- Examples of MAC systems include SELinux (Security-Enhanced Linux) and AppArmor.

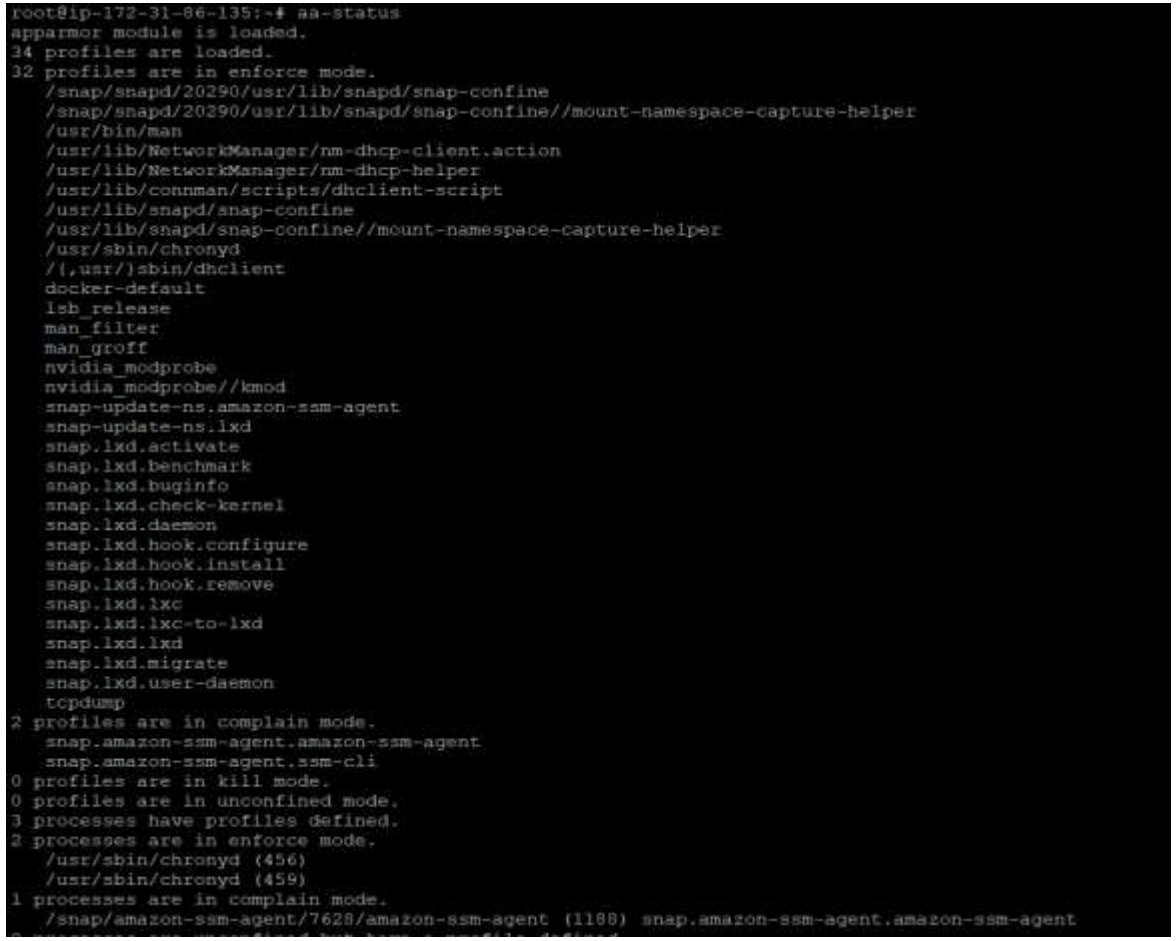
DAC (Discretionary Access Control)

- each user has control over the access permissions of their own resources. Access control decisions are at the discretion of the resource owner.
- DAC commonly uses Access Control Lists (ACLs) and ownership information to determine access rights. Each resource has an owner, and the owner can set permissions.
- Unix and Linux systems often use DAC. Permissions on files and directories (read, write, execute) are controlled by the file owner, group, and others.

4. Mandatory Access Control Systems: AppArmor

AppArmor implements its controls by defining different profiles that can be applied to processes running on the host. These profiles can restrict access to a number of resources, including files, network traffic, and Linux capabilities.

aa-status



4. Mandatory Access Control Systems: AppArmor

AppArmor is enabled for docker containers by default.

There is one profile docker-default entry. This profile is provided by default in Docker and is designed to offer some protection without risking application compatibility.

docker run –dt nginx

As soon as we start a container, docker-default profile applied in enforced mode

```
0 profiles are in kill mode.
0 profiles are in unconfined mode.
6 processes have profiles defined.
5 processes are in enforce mode.
   /usr/sbin/chronyd (456)
   /usr/sbin/chronyd (459)
   /usr/sbin/nginx (16671) docker-default
   /usr/sbin/nginx (16722) docker-default
   /usr/sbin/nginx (16723) docker-default
1 processes are in complain mode.
```

4. Mandatory Access Control Systems: AppArmor

Parsing AppArmor Profiles

AppArmor allows a number of options using apparmor_parser to parse either its default or custom generated profiles. apparmor_parser is widely used to load, unload, debug, remove, replace, cache and match-strings within profiles out of the other available options.

-a – Default Action to load a new profile in enforce mode.

-C – Loading a new profile in complain mode.

-r – Overwrite an existing profile.

-R – Remove an existing profile in the kernel.

-V – Display the profile version.

-h – Display reference guide.

4. Mandatory Access Control Systems: AppArmor

Custom profile for AppArmor:

Create a dir and file	**/etc/apparmor.d/containers/docker-block-bin**

Paste the rule	#include <tunables/global>
	profile docker-block-bin flags=(attach_disconnected, mediate_deleted) {
	  #include <abstractions/base>
	  file,
	  deny /etc/** wl,
	}

Load into the Kernel	sudo apparmor_parser -r /etc/apparmor.d/containers/docker-block-bin
Start a container	docker run --rm -it --name web **--security-opt "apparmor=docker-block-bin"** ubuntu /bin/bash

4. Mandatory Access Control Systems: AppArmor

Another Profile:

vi /etc/apparmor.d/containers/no-ping

```
#include <tunables/global>

profile no-ping flags=(attach_disconnected,mediate_deleted) {
  #include <abstractions/base>

  network inet tcp,
  network inet udp,
  network inet icmp,

  deny network raw,
  deny network packet,
  file,
  mount,
}
```

```
apparmor_parser -r /etc/apparmor.d/containers/no-ping
```

4. Mandatory Access Control Systems: AppArmor

docker run -it --security-opt apparmor=no-ping cloudsihmar/container-tools

```
root@ip-172-31-86-135:~#
root@ip-172-31-86-135:~# docker run -it --security-opt apparmor=no-ping cloudsihmar/container-tools
root in container ID ce1dfcba4316 in /~
# amicontained
Container Runtime: not-found
Has Namespaces:
        pid: true
        user: false
AppArmor Profile: no-ping (enforce)
Capabilities:
        BOUNDING -> chown dac_override fowner fsetid kill setgid setuid setpcap net_bind_service net_raw sys_chroot mknod audit_write setfcap
Seccomp: filtering
Blocked Syscalls (57):
        SOCKET MSGRCV PTRACE SYSLOG SETSID SETGROUPS USELIB USTAT SYSFS VHANGUP PIVOT_ROOT _SYSCTL ACCT SETTIMEOFDAY MOUNT UMOUNT2 SWAPON SWAPOFF REBOOT SETHOS
TNAME SETDOMAINNAME IOPL IOPERM CREATE_MODULE INIT_MODULE DELETE_MODULE GET_KERNEL_SYMS QUERY_MODULE QUOTACTL NFSSERVCTL GETPMSG PUTPMSG AFS_SYSCALL TUXCALL SE
CURITY LOOKUP_DCOOKIE CLOCK_SETTIME VSERVER MBIND SET_MEMPOLICY GET_MEMPOLICY KEXEC_LOAD ADD_KEY REQUEST_KEY KEYCTL MIGRATE_PAGES UNSHARE MOVE_PAGES PERF_EVENT
_OPEN FANOTIFY_INIT OPEN_BY_HANDLE_AT SETNS KCMP FINIT_MODULE KEXEC_FILE_LOAD BPF USERFAULTFD
Looking for Docker.sock
root in container ID ce1dfcba4316 in /~
# ping 8.8.8.8
ping: icmp open socket: Permission denied
```

4. Mandatory Access Control Systems: AppArmor

Commonly used command options on profile files :

- r – reading data
- w – creating, deleting or write on an existing file
- x – executing a file
- m – memory mapping an executable file

5. Seccomp

- SECCOMP stands for **secure computing** and it is a Linux Kernel Level feature. that can be used to **sandbox applications** to only use the SYSCALLS they need.
- Seccomp filters are a way of restricting which Linux syscalls a process can perform.
- Syscalls are essentially the interface between userspace programs and the Linux kernel.
- Whenever a program needs access to a service provided by the host kernel (e.g., when opening a file or creating a new process), it uses a syscall to get that.
- Linux machines offer a wide range of syscalls—currently, more than **300** are available.
- It's important to note that syscalls vary depending on the underlying hardware architecture. For example, you will see differences in the syscall table between **ARM-based systems and AMD64 ones.**

5. Seccomp

SECCOMP can operate with three modes.

- **Mode 0** implies that SECCOMP is disabled.
- **MODE 1** implies that it is applied in a strict mode which will block all syscalls except for four which is read, write, exit and ciggrette on syscalls.
- **MODE 2** it selectively filter syscalls.

docker has a build in SECCOMP filter that it uses by default whenever we create a container provided that the host kernel has SECCOMP enabled

5. Seccomp

Strace:

This is used to check the syscalls made by any process.

strace -c -f -S name whoami 2>&1 1>/dev/null | tail -n +3 | head -n -2 | awk '{print $(NF)}'
strace -c -f -S name time 2>&1 1>/dev/null | tail -n +3 | head -n -2 | awk '{print $(NF)}'
strace -c -f -S name chmod 2>&1 1>/dev/null | tail -n +3 | head -n -2 | awk '{print $(NF)}'
strace -c -f -S name mkdir 2>&1 1>/dev/null | tail -n +3 | head -n -2 | awk '{print $(NF)}'

5. Seccomp

Demo :

https://github.com/CloudSihmar/seccomp.git

docker run -it ubuntu /bin/bash

grep Seccomp /proc/1/status

```
root@ip-172-31-86-135:~# docker run -it ubuntu /bin/bash
root@8ba98f52f292:/# unshare
unshare: unshare failed: Operation not permitted

root@8ba98f52f292:/# grep Seccomp /proc/1/status
Seccomp:        2
Seccomp_filters:        1
```

5. Seccomp

Applying Mode 0:

docker run -it --rm --security-opt seccomp=unconfined ubuntu

```
root@ip-172-31-86-135:~# docker run -it --rm --security-opt seccomp=unconfined ubuntu
root@036f075f7429:/#
root@036f075f7429:/# unshare
# exit
root@036f075f7429:/# grep Seccomp /proc/1/status
Seccomp:        0
Seccomp_filters:        0
```

5. Seccomp Profiles

Whitelist.json

```
{
  "defaultAction": "SCMP_ACT_ERRNO",
  "architectures": [
    "SCMP_ARCH_X86_64",
    "SCMP_ARCH_X86",
    "SCMP_ARCH_X32"
  ],
  "syscalls": [
  ]
}
```

5. Seccomp Profiles

SECCOMP profile primarily consists of three elements.

- First is **architecture** which defines which system it is defined for.
- Second is Syscall array in which we can define an **array of SYSCALL names** and associated action whether to allow or deny them.
- finally default behavior, when dealing with SYSCALLS that have not been declared inside the SYSCALLS array.

defaultAction: SCMP_ACT_ERRNO

if the SYSCALL needed by the application is not added to the white list, it can cause the application to fail.

5. Seccomp Profiles

Blacklist.json

There is another type of SECCOMP profile which is called as **blacklist**.

- It does the opposite of a whitelist, it allows every SYSCALL by default and rejects those that we have specifically defined inside the SYSCALLS array.

  defaultAction: SCMP_ACT_ALLOW

- They are easier to create a compared to the whitelist, however they are more susceptible to attacks. They allow all syscalls by default, if a potentially dangerous SYSCALL is not added to the blacklist, this can lead to security incident.

5. Seccomp

Demo:

Deny.json

```
{
    "defaultAction": "SCMP_ACT_ERRNO",
    "architectures": [
            "SCMP_ARCH_X86_64",
            "SCMP_ARCH_X86",
            "SCMP_ARCH_X32"
    ],
    "syscalls": [
    ]
}
```

```
docker container run -it --cap-add ALL --security-opt apparmor=unconfined --security-opt seccomp=deny.json ubuntu /bin/bash
```

5. Seccomp

Demo:

Selectively remove syscalls

docker container run --rm -it --security-opt seccomp=default-no-chmod.json ubuntu /bin/bash

```
root@ip-172-31-86-135: docker container run --rm -it --security-opt seccomp=default-no-chmod.json
ubuntu /bin/bash
root@3689b161c7b6:/# chmod 777 / -v
chmod: changing permissions of '/': Operation not permitted
failed to change mode of '/' from 0755 (rwxr-xr-x) to 0777 (rwxrwxrwx)
root@3689b161c7b6:/#
```

5. Seccomp

Demo:

Use the default one

```
docker container run --rm -it --security-opt seccomp=default.json ubuntu /bin/bash
root@c5bf29f882f4:/# chmod /
chmod: missing operand after '/'
Try 'chmod --help' for more information.
root@c5bf29f882f4:/# chmod 777 / -v
mode of '/' changed from 0755 (rwxr-xr-x) to 0777 (rwxrwxrwx)
```

5. Seccomp

Demo:

Allow everything

llow.json

```
{
	"defaultAction": "SCMP_ACT_ALLOW",
	"architectures": [
		"SCMP_ARCH_X86_64",
		"SCMP_ARCH_X86",
		"SCMP_ARCH_X32"
	],
	"syscalls": [
	]
}
```

```
docker container run --rm -it --security-opt seccomp=allow.json ubuntu /bin/bash
```

Seccomp

- Seccomp filters are written **Berkeley Packet Filter (BPF)** programs that restrict which syscalls a process can make, allowing for very fine-grained restrictions to be put in place.
- The most widely used example of this is in containers, where users often run as root and there is a need to block dangerous syscalls that could allow for container breakout.
- As part of the default set of restrictions that Docker puts in place, a seccomp filter is applied to any new container. This filter can catch cases where an action in a container would be allowed by the other layers of protection (e.g., capabilities).
- Historically, there have been several cases where a security exploit would have been blocked by Docker's seccomp filter. One example is **CVE-2022-0185**, which uses the **unshare syscall** to exploit a vulnerability. This syscall is blocked by Docker's seccomp filter.

Docker Recommendations

- Use minimal base images. Bigger the image, the wider the attack surface and potential for vulnerabilities.

For Example: FROM alpine: 3.15

Docker Recommendations

The .dockerignore file is used to specify files and directories that should be excluded when building a Docker image.

It helps in optimizing the build context and preventing unnecessary files from being included in the image. Additionally, it can be used to avoid accidental inclusion of sensitive data.

```
# Exclude sensitive files
.env
sensitive_data/
confidential.txt
```

Docker Recommendations

- Use fixed tags for immutability
- Using fixed tags for immutability in the context of container images refers to assigning a specific, unchanging tag to your Docker images.
- This practice ensures that the tag points to a specific and immutable version of the image. Immutability is essential for reproducibility and helps in avoiding unintended changes when deploying or referencing container images.
- Avoid using the latest tag for production images. The latest tag is not immutable as it always points to the latest build, which can lead to unintended changes.
- For example: image:latest vs image:v1 vs image:v3

Docker Recommendations

- Never use ADD, use COPY
- COPY has a straightforward and predictable behavior. It performs a simple copy operation from the source to the destination without additional features like TAR extraction or URL fetching, which are present in ADD. This simplicity reduces the risk of unexpected behavior.
- For instance, ADD can fetch files from URLs and automatically extract TAR files, which may introduce security concerns if not used carefully.
- The COPY instruction is more cache-efficient. Docker builds can leverage layer caching, and if the source files haven't changed, the build process can use the cached layers, leading to faster builds. ADD is less cache-friendly due to its additional features, which can lead to unnecessary invalidation of the cache.

Docker Recommendations

- Limit runtime options with flags.
- Important flags to consider are :-
- Limit how much memory can be accessed with –memory flag
- Limit how much available CPU resources can be used with –cpus flag
- Limit the file system to be read-only with the –read-only flag
- Ensure “on-failure” container restart policy to set to ‘5’ using the –restart flag

```
docker run --restart=on-failure:5 your-image-name
```

- Ensure the container is restritced from acquiring additional privileges via the –security-opt=“no-new-privileges:true”

```
docker run --security-opt="no-new-privileges:true" image-name
```

Testing Docker

- Test that Dockerfile follows best practices by using a linter:
  - Hadolint (https://github.com/hadolint/hadolint)
  - dockle (https://github.com/goodwithtech/dockle)
- Test Dockerfile base image for vulnerabilities
  - trivy (https://github.com/aquasecurity/trivy)

    Check the vulnerabilities here (https://cve.mitre.org/cve/search_cve_list.html)
  - snyk (https://snyk.io/)
- Reference CIS benchmarks for Docker
- docker-bench-security

Module 10:
IAC Security

Terraform Recommendations

- Never store secrets in plane text
- Use Terraform variables
- Use environment variables for credentials e.g pipeline env vars (Easiest)
- leverage external secret management tools e.g. Azure Key Vault, HashiCorp Vault or AWS Secrets Manager (Harder)

```
data "vault_generic_secret" "aws_credentials" {
  path = "secret/aws"
}

provider "aws" {
  region     = "us-west-2"
  access_key = data.vault_generic_secret.aws_credentials.data["access_key"]
  secret_key = data.vault_generic_secret.aws_credentials.data["secret_key"]
}
```

Terraform Recommendations

- Pin provider version
- This allows easy creation of pipelines that would not auto upgrade and break pipelines stopping users from using the pipeline. Just ensure you schedule regular manual updates to fix breaking changes between versions.

```
provider "aws" {
  region = "us-west-2"
  version = "=2.70.0"  # Pin to a specific version
}
```

Terraform Recommendations

- Ensure Terraform has a dedicated user in cloud provider e.g. AWS or AZURE
- for example: A new role is created in AWS specially for Terraform (not used by anything else) and has the least privilege i.e. minimum permissions required

Testing Terraform

Test code with static analysis

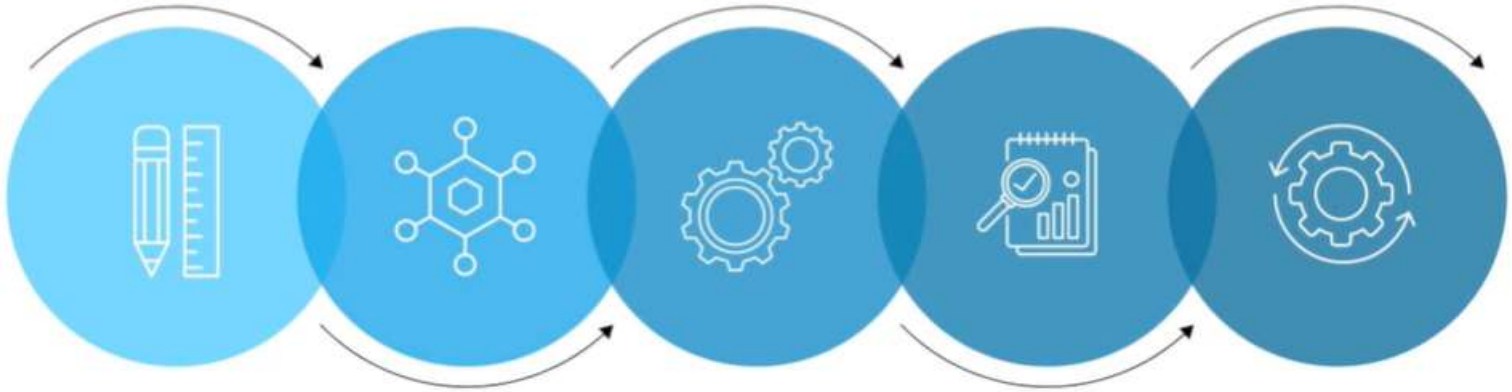
- tfsec (https://github.com/aquasecurity/tfsec)
- Checkov
- terrascan

Module 11:
Incident Management

Incident Management

We know how DevOps has changed how we deliver software. But what about after the deployment?

Whether you are in a traditional operations organisation or a “you build it, you run it” team, how do you mobilise, resolve, and learn from incidents?



Incident Management Activities

1. Detecting and recording incident details

2. Matching incidents against known problems

3. Resolving incidents as quickly as possible

4. Prioritizing incidents in terms of impact and urgency

5. Escalating incidents to other teams to ensure timely resolution

Benefits of DevOps Incident Management Implementation

It is important for any IT department to have a plan for managing incidents. After all, no matter how good you are at predicting events, an incident can still happen. Here are the top four ways implementing incident management will help you.

1. Maintaining SLA’s

2. Meeting Service Availability Requirements

3. Increasing Staff Efficiency and Productivity

4. Improving User Satisfaction

The DevOps incident management process

The DevOps approach to managing incidents isn’t radically different from the traditional steps to effective incident management. DevOps incident management includes an explicit emphasis on involving developer teams from the beginning, including on call and assigning work based on expertise, not job titles.

Best Practices for effective DevOps IM teams

1. Automate processes and workflows

2. Communicate between teams

3. Identify and focus on the business bottom line

4. Use the blameless approach

5. Utilize on-call scheduling to position developers and sysadmins as SREs

Measure Everything

- Deployment Frequency
- Change Volume
- Lead Time from Development to Deployment
- Percentage of failed deployments
- Meantime to recovery MTTR
- Customer Ticket Volume
- Percentage change in User Volume
- High Availability
- Performance – Response time

Module 12:
Compliance and Governance

The significance of compliance in DevSecOps

1. Regulatory Alignment:

Different industries are subject to different rules, such as PCI DSS for the banking industry and HIPAA for the healthcare industry. DevSecOps compliance ensures that security procedures comply with these legal requirements, protecting sensitive data and reducing legal risks.

2. Automated Compliance Checks:

DevSecOps is built on automation. Compliance checks are carried out using automated tools and scripts to ensure consistent execution of security. This proactive strategy finds and fixes security issues as they arise, preventing them from worsening.

The significance of compliance in DevSecOps

3. Collaboration Across Teams:

Collaboration is the vitality of DevSecOps. Effective communication and cooperation across the development, security, and operations teams are essential to compliance initiatives. This collaboration culture ensures that security needs are well-understood and seamlessly incorporated across all teams.

4. Continuous Monitoring:

Due to the dynamic nature of compliance, security controls must be continuously monitored. DevSecOps uses automated technologies to continually evaluate and address security flaws and compliance violations throughout the development lifecycle.

The significance of compliance in DevSecOps

5. Security as Code:

According to the DevSecOps framework, security is a collection of configurations, rules, and regulations maintained in code repositories. This approach makes possible version control, automated testing, and seamless interaction with the broader development process.

6. Audit Trails and Documentation:

Complete documentation is essential to compliance. Teams working on DevSecOps keep detailed logs of all changes and security measures. This paperwork acts as an audit trail to demonstrate conformity to security standards during compliance audits.

The significance of compliance in DevSecOps

7. Risk Mitigation and Reputation Management:

DevSecOps compliance is essential for reducing security risks. Adhering to standards and best practices aids in preventative vulnerability management and protecting data and reputation. In the event of an incident, adherence to compliance serves as a buffer, limiting the adverse legal and financial effects. Fortifying defenses and establishing confidence in software development, DevSecOps compliance is not just a necessity but a strategic imperative when integrated effortlessly.

Benefit of Achieving Compliance in DevSecOps

1. Risk Reduction:

Achieving compliance in DevSecOps finds and fixes security flaws, lowering the risk of incidents and possible breaches. This proactive approach protects sensitive data and intellectual property while reducing the effects of security risks.

2. Operational Efficiency:

Compliance integration helps to improve DevSecOps workflow operational efficiency. The detection and resolution of security concerns are made more accessible by automated compliance checks and continuous monitoring systems. This effectiveness speeds up the development process and ensures that security controls are continuously used across the whole software development lifecycle.

Benefit of Achieving Compliance in DevSecOps

3. Collaborative Security Culture:

Achieving compliance helps the organization develop a collaborative security culture. Teams in development, security, and operations should work together more effectively by removing barriers. Due to the seamless integration of security practices made possible by this collaborative mindset, the development process is more safe and robust.

4. Competitive Advantage:

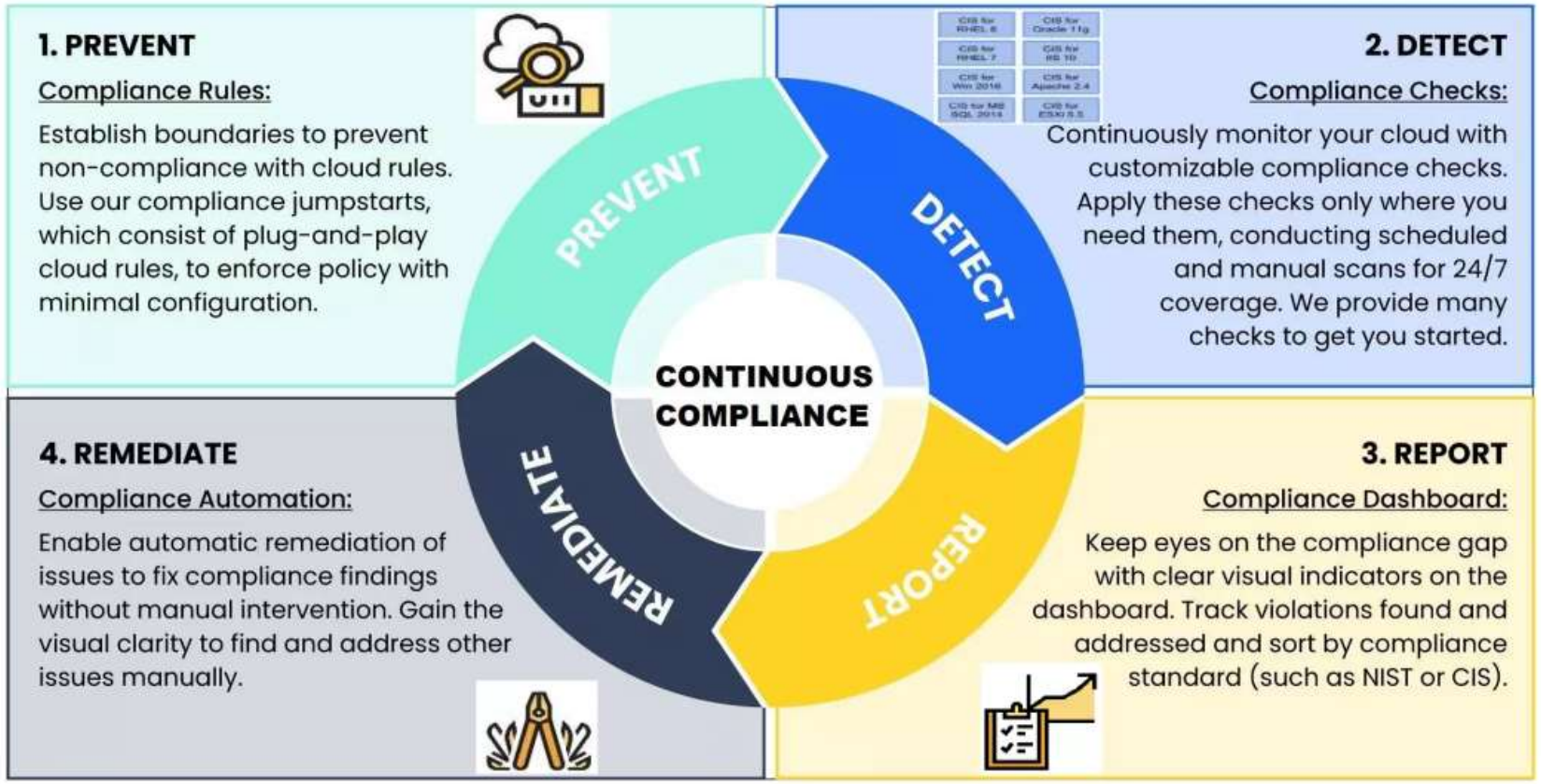
Organizations that successfully implement DevSecOps compliance enjoy a competitive edge. It distinguishes them from competitors in the market and assures clients and partners of the effectiveness of the security measures implemented. This competitive advantage may be crucial for landing contracts and building stakeholder trust.

Benefit of Achieving Compliance in DevSecOps

5. Legal and Financial Safeguards:

Legal and financial safeguards are provided by following compliance requirements, which act as a barrier in the event of a security problem. As a legal and financial protection, regulatory agencies frequently consider an organization’s compliance efforts when deciding penalties. Legal obligations and related cost implications might be reduced with a well-established compliance structure.

Best Practices of Continuous Compliance



Tools to implement continuous Compliance

1. Compliance Monitoring and Dashboard tools (e.g.: Puppet CIS Compliance, Chef Inspec, Upguard, ZenGRC, IBM Powertech, etc.)
2. Process Mining Tools (e.g.: Signavio, Minit, UiPath, etc.)
3. Infrastructure as Code automation tools (e.g.: Puppet, Ansible, Chef, etc.)
4. AIOps tools (e.g.: Devo, Broadcom AIOPS, SumoLogic, Moogsoft, etc.)
5. All of the above

Module 13: ArgoCD

What is ArgoCD?

- ArgoCD is an open-source declarative continuous delivery tool for Kubernetes applications.
- It is part of the Argo Project, which includes several tools designed to simplify and automate various aspects of Kubernetes workflows.
- ArgoCD specifically focuses on deploying and managing applications in Kubernetes clusters while ensuring they adhere to a desired state defined in a Git repository.

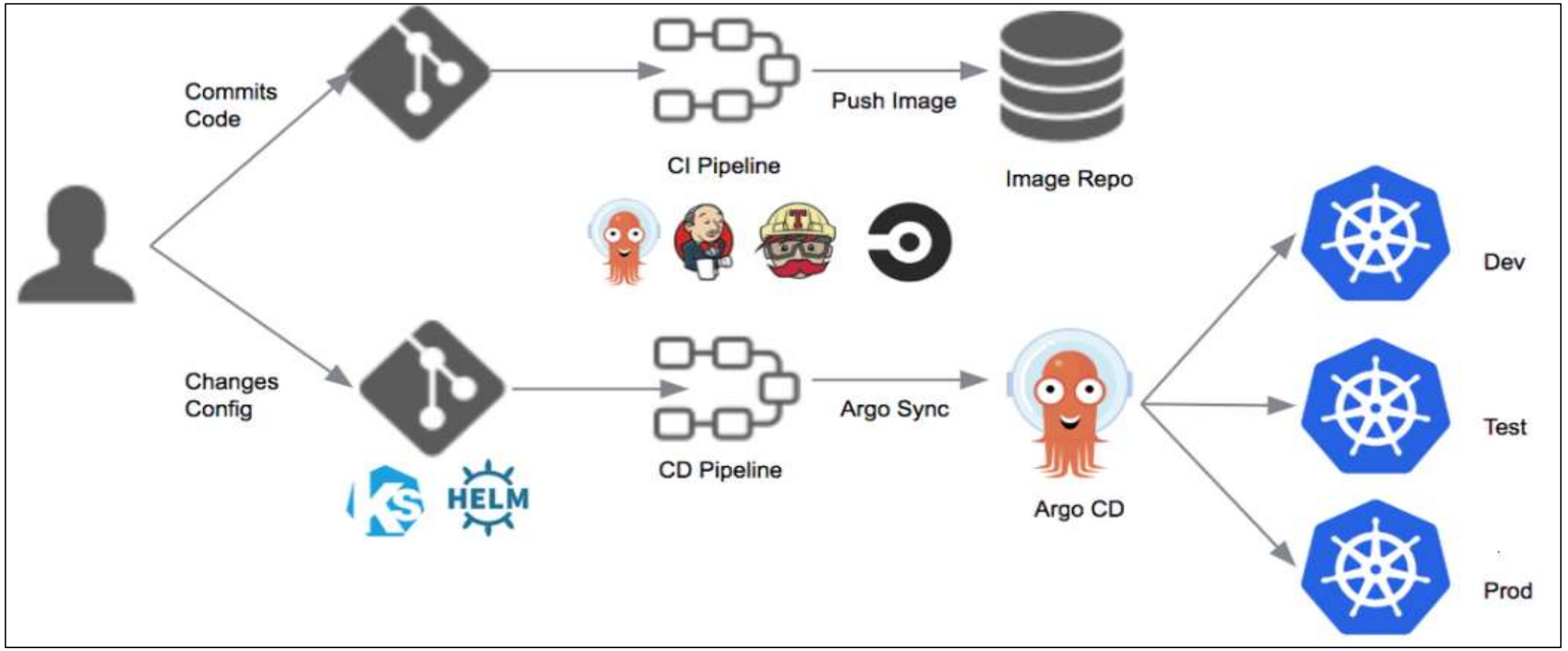
Challenges with other Tools

- Install and setup tools like Kubectl
- Configure access to Kubernetes
- Configure access to cloud platforms
- Security challenge
- No visibility of deployment status

ArgoCD as a better tool

- ArgoCD is part of Kubernetes Cluster
- ArgoCD agent pulls k8s manifest changes and applies them

CD workflow with ArgoCD



CD workflow with ArgoCD

- Deploy ArgoCD in K8s cluster
- Configure ArgoCD to track Git repository
- ArgoCD monitors for any changes and applies automatically

CD workflow with ArgoCD

- Separate git repository for application source code and application configuration (K8s manifest files)
- Even separate git repository for system configurations
- Manifest files are independent of source code.
- CI tool updates Deployment.yaml in separate git repo.
- ArgoCD supports Kubernetes YAML files, Helm Charts and Kustomize files
- Git repository is tracked and synced by Argo CD.
- Separate pipelines for CI and CD.
- CI is handled by developers and devOps members whereas CD is handled by Operations and DevOps people.

Benefits of ArgoCD

- Whole K8s configuration defined as Code in Git Repository
- Config Files are not applied manually from local laptops
- Same Interface for updating the cluster
- ArgoCD watches the changes in the cluster as well
- ArgoCD compares desired configuration in the Git repo with the actual state in the K8s cluster
- ArgoCD will sync the changes, overwriting the manual change.
- ArgoCD guarantees that K8s manifest in Git remains single source of truth

Benefits of ArgoCD

- Full cluster transparency
- If we want to manually change in the cluster, then configure ArgoCD to not sync manual cluster changes automatically
- We will get history of changes and version controlled changes.
- Better team collaboration
- Easy rollback
- Cluster Disaster Recovery (using the complete cluster configuration file in Git repo)
- Manage cluster access indirectly via Git
- No need to create CLusterRole and User resources in Kubernetes
- No cluster credentials outside of K8s

ArgoCD as Kubernetes extension

- ArgoCD uses existing K8s functionalities
- It is using etcd to store data
- It uses K8s controllers for monitoring and comparing actual and desired state
- It monitors the real-time updates of application state

ArgoCD configuration

Install ArgoCD in K8s cluster
kubectl create namespace argocd kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml

- Change the argocd-server svc from ClusterIP to NodePort
- Get the secret argocd-initial-admin-secret and decode it.

  echo N0lzTFpsNlhvdzhqNVZDSA== | base64 --decode

ArgoCD configuration

Create an application resource to sync with the repo

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: myapp
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/CloudSihmar/argocd-example.git
    targetRevision: HEAD
    path: .
  destination:
    server: https://kubernetes.default.svc
    namespace: myapp
  syncPolicy:
    syncOptions:
    - CreateNamespace=true
    automated:
      selfHeal: true
      prune: true
```