|  |  |
| --- | --- |
| Name: | Arpitha S P |
| Lab User ID: | 23SEK3324\_U10 |
| Date: | 09-01-2024 |
| Application Name: | Vulnerable Java Web Application |

**Follow the below guidelines:**





System Architecture:

(Understand the system and document the physical and logical architecture of the system, use the shapes and icons to capture the system architecture)

EC2 Instance-VM

13.234.19.138:8000

Docker Engine

Container

Docker Image

Physical Architecture :

* EC2 Instance
  + Instance type : t2 medium
  + Platform : ubuntu (Linux/UNIX)
  + Volume size(GiB) : 20
  + Availability zone : ap-south-1a

Logical Architecture :

* Docker Containers : These are isolated instances running within docker engine
* Docker Engine manages and orchestrates these containers

Define system’s normal behavior:

(Define the steady state of the system is defined, thereby defining some measurable outputs which can indicate the system’s normal behavior)

* Handling Requests: When user interacts with application by entering the url, submitting a form or clicking the links, the web server receives a request and passes them to java application running on the server
* The java application processes these requests which may involve various actions like authentication, data retrieval from db or file executions.
* Based on the users input the application generates dynamic content.
* Vulnerabilities like SQL Injection occur when user input isn’t properly sanitized, allowing attackers to inject malicious SQL queries, potentially accessing or modifying the database.

Hypothesis:

(During an experiment, we need a hypothesis for comparing to a stable control group, and the same applies here too. If there is a reasonable expectation for a particular action according to which we will change the steady state of a system, then the first thing to do is to fix the system so that we accommodate for the action that will potentially have that effect on the system. For eg: "If one of our database servers fails, our service will automatically switch to a backup server, and users will not experience any downtime or data loss.")



Well-documented vulnerabilities within java web applications that are widely recognized and understood.

Eg : SQL Injection

Vulnerabilities suspected to exist but not fully identified or comprehensively addressed. These are potential weaknesses that haven’t been fully uncovered.

Eg : Emerging security threats in specific libraries.

**Known**

Unrecognized vulnerabilities within an application or system, overlooked by those responsible for its security. These vulnerabilities exist but are not actively identified.

Eg : Unintentional logical flaws in authentication.

**Unknown**

Completely unexpected vulnerabilities that are not considered within current knowledge. These are unforeseen issues that arise unexpectedly.

Eg : Entirely new attack vectors.

**Unknown**

**Known**

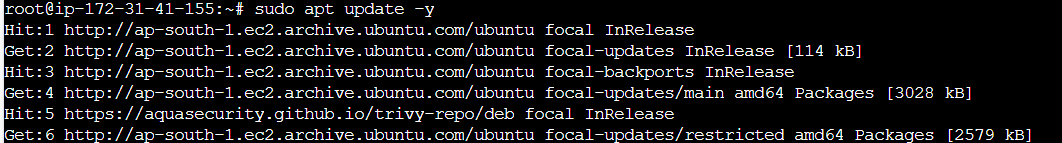
Experiment:

(Document your Preparation, Implementation, Observation and Analysis )

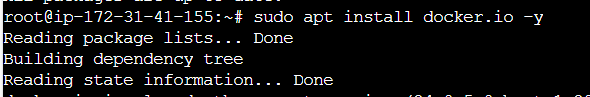
This repository contains a sample application, the "Websites Tester Service", that's vulnerable to a Command

Injection and Server-Side Request Forgery (SSRF) vulnerability.

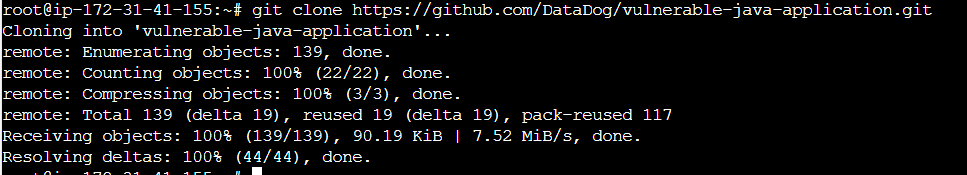
Step 1 : Updating the packages



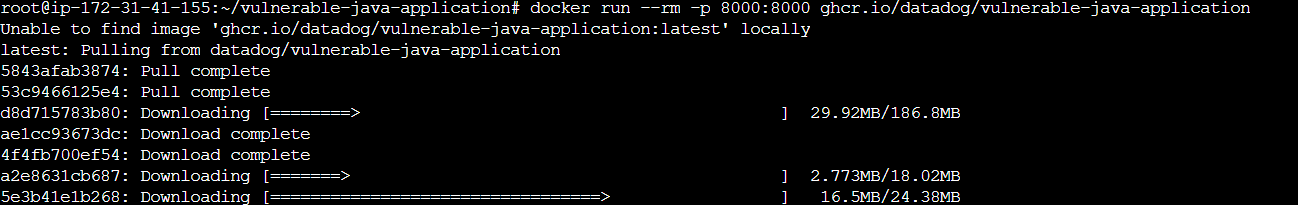
Step 2 : Installing Docker

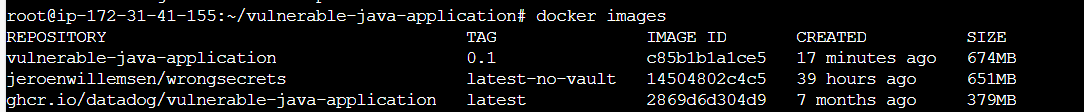


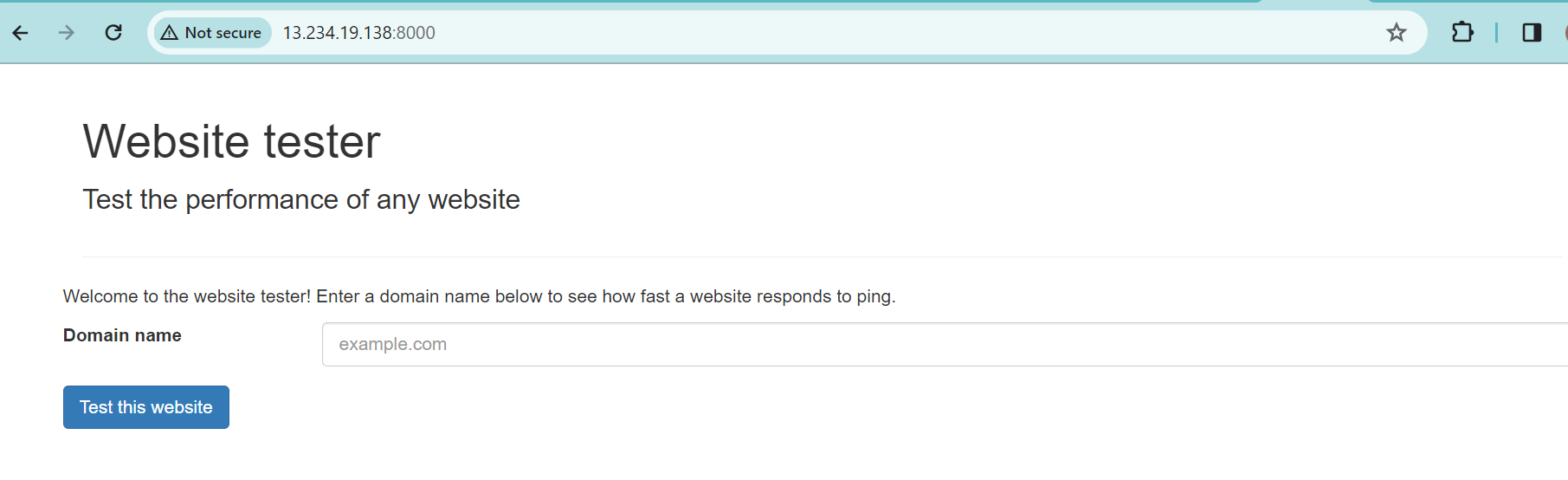
Step 3 : Clone Repository



Step 4 : Docker Run

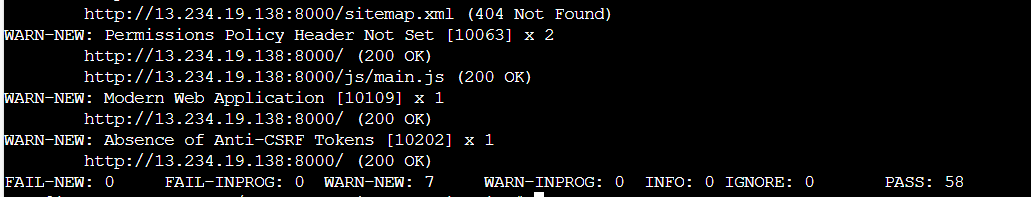




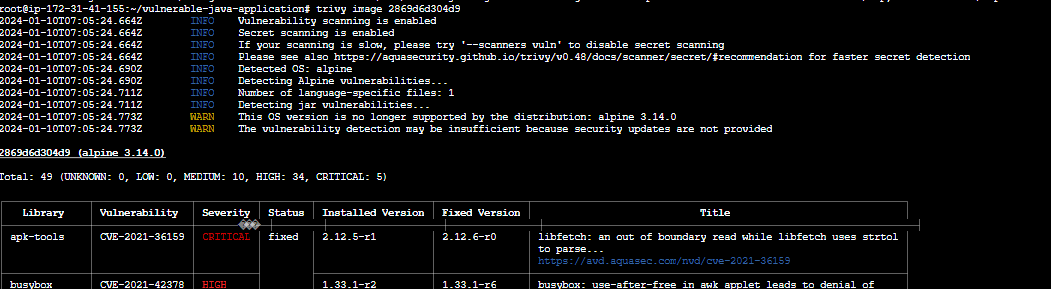


**Vulnerability Scan using ZAP:**





**Vulnerability Scan using Trivy:**





**Docker :**

Docker is a platform designed to make it easier to create, deploy, and run applications using containers. It provides

tools and platform to manage these containers, making it simpler to build and run applications consistently across

different environments.

**Docker run :** docker run is used to create and start a container based on a specific docker image.

Docker run –rm -p 8000:8000 ghcr.io/datadog/vulnerable-java-application: runs the docker container using the image named ghcr.io/datadog/vulnerable-java-application.

**‘--rm’** : removes the container automatically after it stops running. Keeps your system clean from unused containers

**‘-p 8000:8000’** : port mapping . It maps port 8000 of the host machine to port 8000 of the container.

**ZAP(Zed Attack Proxy) :**

It is an open-source web application scanner developed by OWASP(Open web application Security Project).

ZAP scans web applications to identify security vulnerabilities such as SQL Injection and more.

1. Permission policy header not set [10063] :

It is basically a security feature in a web browser that help prevent certain types of attacks. We are getting this error which means, the website is missing this security header or its not properly configured. Permissions Policy provides a set of standard HTTP headers that allow website owners to limit which features of browsers can be used by the page such as camera, microphone, location, full screen etc.

Solution : Its advisable to implement and configure the “Permission-policy” HTTP header to ensure proper security measures are in place.

1. Absence of Anti-CSRF Tokens:

They are used to prevent CSRF attacks, where unauthorized commands are transmitted from user that the web application trusts. These tokens are added to forms or requests and verified by the server to ensure that the request comes from an authenticated and authorized user. If a web application lacks these tokens, it becomes vulnerable to CSRF attacks, potentially allowing to perform actions on behalf of authenticated users without their consent.

Solution : Utilize built in functionalities provided by many web fameworks/ libraries for CSRF protection.

Create unique tokens for each user session and embed them in forms or requests so the server should validate these tokens to ensure request is originated from expected user.

Tie CSRF tokens to secure cookies to ensure they’re transmitted and validated correctly with each request.

**Trivy :**

Trivy is an open-source vulnerability scanner primarily used for container security. Its designed to identify vulnerabilities within container images. It enables organizations to mitigate potential risks before deploying their applications into production environment.

1. CVE-2021-36159 (out-of-bounds read)

The product reads data past the end, or before the beginning, of the intended buffer. The FTP passive mode implementation allows an out-of-bounds read because strtol is used to parse the relevant numbers into address bytes. It does not check if the line ends prematurely. If it does, the for-loop condition checks for the 0 terminator one byte too late.

Solution : Assume all input is malicious. Use an “accept known good” input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications or transform it into something that does. To reduce the likelihood of getting out-of-bound read, we should ensure to validate and correct calculations for any length argument, buffer size calculation, or offset.

1. CVE-2021-42378(Use After Free)

Referencing memory after it has been freed can cause a program to crash, use unexpected values, or execute code. A use-after-free in Busyboxs awk applet leads to denial of service and possibly code execution when processing a crafted awk pattern in the getvar\_i function. the memory in question is allocated to another pointer validly at some point after it has been freed. The original pointer to the freed memory is used again and points to somewhere within the new allocation. As the data is changed, it corrupts the validly used memory.

Solution: Ensure we are using the latest patched version of busybox. Review the code for the awk applet in BusyBox to identify where the memory is being properly accessed after being freed. Ensure proper memory management practices are followed. Apply patches or fixes to the code to resolve the use-after-free vulnerability.

1. CVE-2022-0778(Loop with Unreachable Exit Condition ('Infinite Loop')):

This particular vulnerability refers to a loop with an unreachable exit condition, leading to an infinite loop. It’s an issue that can cause a program to get struck indefinitely, consuming system resources and potentially affecting system stability or availability.

Solution: Review the affected code to identify the loop causing the infinite loop condition. Ensure that lopps have proper and reachable exit conditions, preventing them from entering infinite iterations.

1. CVE-2022-37434(Out-of-bounds Write):

The product writes data past the end, or before the beginning, of the intended buffer. This could lead to adjacent memory,crashes by attackers.

Solution: Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Run or compile the software using features or extensions that randomly arrange the positions of a program’s executable and libraries in memory. Because this makes the addresses unpredictable, it can prevent an attacker from reliably jumping to exploitable code.