|  |  |
| --- | --- |
| Name: | Ajay Jain |
| Lab User ID: | 23SEK3324\_U33 |
| Date: | 10/01/2024 |
| Application Name: | DVWPS |

**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)



Hosted application using wordpress along with my sql on port: 31337

**Ubuntu 20.04** Created a wordpress container with dvwps as image



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)

Basic functionality such as login, signup works perfectly.

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.")



**Known**

If there is any traffic then functionalities might get affected but don’t know how

Login Signup is working fine

**Unknown**

**Unknown**

**Known**

Any unknown vulnerabilities in code that might affect the availability of the application

If there is an increase in traffic then it would affect the functionality but don’t know how swiftly would it perform

Experiment:

(Document your Preparation, Implementation, Observation and Analysis )

**Implementation:**

* Launched EC2 instance on AWS with Ubuntu 20.4
* updated the machine and installed docker
* created a directory where I can clone the vulnerable repository and cloned the dvwps repo in that directory.
* Upload the repository on Snyk to view the vulnerabilities in the webpage.

**Observation and Results:**

1. Upon using Snyk tool we find several Vulnerabilities. Some vulnerabilities are listed below.

**Vulnerabilities Found:**

1. **Vul : sudo Off-by-one Error**

**Impact**: leads to**an unpredictable behavior of application,** depending on**nature** of the vulnerability, and in most cases results in application crash or infinite loop.

**Fix**: Upgrade to sudo@1.8.9p5-1ubuntu1.5+esm6

1. **Vul : expat/libexpat1-Exposure of Resource to Wrong Sphere**

**Impact:** Resources such as files and directories may be inadvertently exposed through mechanisms such as insecure permissions, or when a program accidentally operates on the wrong object. For example, a program may intend that private files can only be provided to a specific user. This effectively defines a control sphere that is intended to prevent attackers from accessing these private files. If the file permissions are insecure, then parties other than the user will be able to access those files.

**Fix**: Use proper access control mechanisms to restrict who can access your resources, and enforce the principle of least privilege

1. **Vul : cyrus-sasl2/libsasl2-modules-db SQL Injection**

**Impact:** SQL injection allows the attacker to read, change, or delete sensitive data as well as execute administrative operations on the database.

**Fix**: Prevent user-supplied input from affecting the logic of the executed query. Using parameterized queries which almost all SQL libraries support. In a parameterized query, the input is "escaped" and handled literally.

1. **Vul : Null dereference-The dangers of accessing null objects**

**Impact:** it can cause the program to crash or behave unexpectedly, potentially leading to security issues.

**Fix**: have the requests.extract\_json().then().get() handle the JSON values instead of passing a pointer to that function.

1. **Vul : Cross-site Scripting (XSS)**

**Impact:** XSS allows hackers to inject malicious JavaScript into a web application. Such injections are extremely dangerous from the security perspective, and can lead to:

* Stealing sensitive information, including session tokens, cookies or user credentials
* Injecting multiple types of malware (e.g. worms) into the website
* Changing the website appearance to trick users into performing undesirable actions

**Fix**: The first step to mitigate XSS is to find all places in your code where this pattern occurs. Then the most important XSS mitigation step is to escape your HTML output. To do that, you should HTML-encode all dangerous characters in the user-controlled data before injecting that data into your HTML output.

