Q-1:

Denial of Service (DoS attack) is a cyber attack that aims to temporarily or indefinitely disrupt the services of an internet-connected host, making a machine or network resources inaccessible to actual users. It is usually performed in the form of making the target system unable to respond to incoming requests due to overload due to overloading the target machine or resource with unnecessary requests. The attackers can use amplification techniques for attacks with higher bandwidth than they have and the DNS Protocol statistically has been used/exploited much more than the other protocols.

**a. Why is this type of attack preferred, especially in the DNS protocol? Please explain briefly.\***

The reasons Why DNS protocol is most preferred for a DDOS attack vector;

* DNS is UDP based protocol and UDP protocols don’t have any handshaking dialogues before establishing a session.
* DNS is a mandatory protocol in order to make Internet based resources more user friendly.
* DNS makes a great weapon in producing the massive amounts of traffic that can take down entire networks. DNS Reflection provides up to 80:1 amplification factor for Amplification attacks.
* DNS is easy to launch: The protocol is easily accessible. It doesn’t take a lot of compute to generate a lot of DNS data.
* DNS is easy to hide/spoof: There are tons of open resolvers on the internet that can recursive traffic to a target. Source addresses can easily be spoofed, which contributes to the many cases of DNS attacks.
* DNS is usually unmonitored: Most security teams don’t monitor DNS for potential security issues. Most see the protocol as so old that they simply take it for granted. It’s also just supposed to work.

**b. In DNS Amplification Attacks, are there any technical limitations or disadvantages for the attacker?\***

* Attackers frequently use ANY query type to increase the amplification ratio, making it the most used query type in observed DNS Amplification attacks. However most of the firewalls drop the ANY queries and spoofed packets.
* Attacker would have to have multiple DNS clients and a single attacker may not have sufficient uplink bandwidth to send requests to many amplifiers at the same time. Typically, this would include a botnet, access to dozens or hundreds of DNS resolvers that are capable of amplifying the attack.
* The key to the attack is the specially built attacker DNS server that is authoritative for a domain that the attacker owns.
* Response Rate Limiting is a mechanism for limiting the amount of unique responses returned by a DNS server. This can limit the effectiveness of a DNS amplification attack by dropping responses that exceed the configured rate limit.
* Traditional DNS packets are limited to a maximum length of 512 bytes at the application layer. However, the extension mechanisms for DNS (EDNS) allow larger DNS packets if supported by both the resolver and authoritative server.

**c.Please find a suitable IP address for amplification attacks in DNS protocol and explain step by step how to find such servers easily.**

* Shodan can be used to find available Recursive DNS server (Resolvers) whose recursion is enabled by searching “Recursion: enabled port:53” text. This query retrieves Recursive DNS servers’ information all around the world.
* Select DNS servers from the list and measure the adoption of Query Rate Limiting sending repeated queries to the servers to determine whether the domain employed query rate limiting.
* Find an Internet domain that is registered with many DNS records. During the attack, the DNS queries that request the entire list of DNS records for that domain are sent.

Question 2

The source files of a sample application are attached as a .zip file. Please unzip the zip file  
in a directory. The code in the app.py file in the unzipped directory contains several OWASP  
Top 10:2021 vulnerabilities. Please review the code, identify contained vulnerabilities, and  
suggest possible mitigations to resolve the identified vulnerabilities. Each area has to  
include:  
  
● What is the vulnerability?  
● Why does the vulnerability arise (what is the reason for the vulnerability)?  
● What is your mitigation suggestion to close the vulnerability?  
  
Note 1: 10 areas are given below to write your answers. However, the application may or  
may not include 10 vulnerabilities.  
  
Note 2: Applicants can run the sample application using docker with the “docker-compose  
up” command if necessary.

Vulnerability 1 & Mitigation:\*

Vulnerability: Injection(SQL) (OWASP 2021/3)

SQL Query that is vulnerable to SQL Injection has been used in the application: SQL Injection ("SELECT username,password from users where username = '"+username+"' and password='"+password+"'")

Mitigations:

* Use of parameterized queries, specifying placeholders for parameters so that they are not considered as a part of the SQL command; rather, as solely data by the database
* input sanitization and validation
* Use of Properly Constructed Stored Procedures
* Escaping All User Supplied Input
* Avoiding administrative privileges
* Implementing Web application firewall

Vulnerability 2 & Mitigation:\*

Vulnerability: Cryptographic Failures (Insufficient Cryptography) (DES) (OWASP 2021/2)

* DES (Data Encryption Standard) is no longer be considered secure, its key length of 56 bits makes it now easily broken with modern computers. DES crypto algorithm has been used for token encryption.
* Certain functions, such as MD5, SHA1, and PKCS number 1 v1.5, either have insufficient entropy - meaning a modern computer can easily crack them in a reasonable amount of time - or have known exploits that make them insufficient for encryption today.

Mitigation:

* More secure crypto such as the AES (Advanced Encryption Standard) family of functions with longer key lengths is still considered secure needs to be implemented.

Vulnerability 3 & Mitigation:\*

Vulnerability: Injection (XSS) (OWASP 2021/3)

The value of the “username” variable which is neither sanitized nor filtered information has been used in HTM code. (message = "<html><body><h1>)Error occured during login process of user "+username+"</h1><p>Detail:" + str(e)+ "

Mitigations:

* input sanitization, validation and filtering
* Escaping All User Supplied Input
* Implementing Web application firewall

Vulnerability 4 & Mitigation:\*

Vulnerability: Identification and Authentication Failures (OWASP 2021/7)

There is no check for Identification and Application failures in the application code and the application permits;

* automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.
* brute force or other automated attacks.
* default, weak, or well-known passwords, such as "Password1" or "admin/admin"

Mitigations:

* Add a mechanism to prevent increasingly delay failed login attempts
* Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
* Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session identifier should not be in the URL, be securely stored, and invalidated after logout, idle, and absolute timeouts.
* Add a mechanism that prevents default, weak, or well-known passwords, such as "Password1" or "admin/admin"

Vulnerability: Security Misconfiguration (OWASP 2021/5)

Database connection is created with plain text of username and password and root user who has highest level privilege in the data base is used (user="root", password="p@ssw0rd1").

Mitigations:

* Do not grant any more privileges than necessary. Never grant privileges to all hosts.
* Connect to the database using a different username than the one you use for administrative purposes. Do not give your applications any more access privileges than they need.
* Create a database user who is authorized only for the “assignment” database.
* Username and password information should not be included in a configuration file or a properties file in clear text as this will allow anyone who can read the file access to the resource. If possible, encrypt this information