Question 1:
Denial of Service (DoS attack) is a cyberattack 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.
1. Why is this type of attack preferred, especially in the DNS protocol? Please explain
briefly.
1. Ease of IP Spoofing: IP spoofing is the creation of Internet Protocol (IP) packets that
have a modified source address in order to either hide the identity of the sender,
impersonate another computer system or both.
1. Amplification Factor: A typical DNS request (just a few lines of text) is very
small—usually in the tens of bytes—and returns a response that’s only slightly larger.
A 10-byte DNS request could generate a response that’s 10, 20, or even 50 times
larger.
1. Public DNS Servers: There are a huge number of public DNS servers on the Internet.
The high number of these servers is another reason why the attack is preferred.
1. In DNS Amplification Attacks, are there any technical limitations or disadvantages for
the attacker?
1. DNS Packet Limit: If the DNS packet limit is exceeded, the traffic switches to the
TCP. This is a serious problem and limitation for an attacker.
DNS operations, for example, queries and zone maintenance operations by default use port
53. For performance reasons, queries use the UDP protocol with a block-size limit of 512
bytes. TCP can be optionally negotiated on a transaction-by-transaction basis for query
operations, but due to the performance overhead incurred with TCP, this is essentially a
theoretical capability. Historically, exceeding the 512-byte response size limit was typically
avoided at all costs, and indeed the limit of 13 IPv4 root servers was the maximum that could
be returned in a single 512-byte UDP transaction. (Ron Aitchison - Pro DNS and BIND 10 -

2011)
1. IP Spoofing Enabled Line: An internet connection with IP spoofing is required for this
attack. If the ISP blocks IP spoofing, the attack fails, and the packets cannot reach
the destination.
2. Others:
a. Target Protection Technologies
b. Botnet Usage & Size
1. Please find a suitable IP address for amplification attacks in DNS protocol and
explain step by step how to find such servers easily.
Step-1: Find Public DNS Server(s). - There are many resources on the Internet that host lists
or Shodan can be used.
Step-2: Find domain name(s) whose DNS query response will be large enough. (Remember
512 Byte Limit) - There are many resources on the Internet that host lists.
Step-3: Perform the DNS query with the "dig" tool and make sure the response returned is
large enough. (Example dig output line: ;; MSG SIZE rcvd: 465)
This Public DNS Server, Domain name, and Query type can be used for attack. Finally, for
more servers, domains, and queries, repeat the steps.
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.
Answers are in the following format:
Each answer is evaluated in 3 parts:
● Vulnerability,
● Why does it arise?
● What is the mitigation suggestion?
Answers
==========
● Vulnerability : SQL Injection (2 Points)

● Why does it arise? : Username and password parameter is unsafely concatenated to
SQL query on app.py line 22. (5 Points)
● What is the mitigation suggestion? : Use of parameterized queries should be applied.
(3 Points)
Notes = Some attendees claimed line 48 is also vulnerable to SQL injection. Although the
SQL query is built the same way as line 22, the token parameter is created by the app on
line 47 and does not include any user-supplied input. Therefore the "token" parameter
cannot be controlled to exploit SQL Injection vulnerability.
==========
● Vulnerability: Insecure Token Handling (2 Points)
● Why does it arise? : Token is generated in order to use as a component of the
cookie; however, on the function "token_update," generated token is saved to the
database without assigning it to any user. Function "token_check" only checks if the
token was generated by the application and returns the result accordingly. This may
lead to a way that users can craft new cookies for another user by using a token that
is received during his/her authentication. (5 Points)
● What is the mitigation suggestion? : Design should be reimplemented in a way that
tokens could not be used by another user. Proper session management should be
implemented (3 Points)
Notes = Attendees who stated only lack of token invalidation received 3 points.
==========
● Vulnerability: Cross-Site Scripting (2 Points)
● Why does it arise? : On line 41, the application puts the user-supplied "username"
parameter without output encoding and returns it as a response when an exception
occurs, thus leading to XSS vulnerability. (5 Points)
● What is the mitigation suggestion? : Output encoding should be applied, and inputs
should be sanitized (3 Points)
Notes = Some attendees marked line 101 as well. These attendees take 3 points as a
bonus.
==========
● Vulnerability: Sensitive Information Submitted Using Get Method (1 Point)
● Why does it arise? : Function "login" receives the "password" parameter with HTTP
get request. Which may lead to the disclosure of sensitive information. (2 Points)

● What is the mitigation suggestion? HTTP Post Method should be used in order to
transfer sensitive data. (2 Points)
==========
● Vulnerability : Insufficient Cryptography (1 Points)
● Why does it arise? : Application uses hash/encryption algorithms (md5 - des) or
encryption purposes. Within des encryption, the same weak salt value for every
encryption process and weak key creation mechanism is used in every round. (2
Points)
● What is the mitigation suggestion? : AES 128 and 256 should be used with sufficient
IV, which is created as random and unpredictable. Weak hash/encryption algorithms
should not be used, such as MD5, RC4, DES, Blowfish, SHA1, or two-key triple DES.
(2 points)
Notes = Applicants who mentioned outdated pycrypto library received 3 points.
==========
● Vulnerability: Hardcoded Password/Key Usage (1 Point)
● Why does it arise? : On lines 17, 94, 65, and 77 in app.py application has hardcoded
passwords/keys which are used for database connection and encryption. (2 Points)
● What is the mitigation suggestion? : Application should store passwords and keys
outside of the code in a strongly-protected, encrypted configuration file or database
that is protected from access by all outsiders, including other local users on the same
system. (2 Points)
==========
● Vulnerability: Weak Password Usage (1 Point)
● Why does it arise? : In the App.py file, it can be seen that the application uses weak
passwords to access the database. (2 Points)
● What is the mitigation suggestion? : Password complexity should be met with
security requirements.
● At least 10 characters.
● At most 128 characters
● At least 1 uppercase character (A-Z)
● At least 1 lowercase character (a-z)
● At least 1 digit (0-9)
● At least 1 special character.
● Identical characters in a row must be forbidden. (2 Points)
==========
● Vulnerability: Violation of the least privilege principle (1 Point)
● Why does it arise? : Application uses a “root” user for database connection and query

execution. (Line 16, 93) Which is not suitable for the least privilege principle. (2
Points)
● What is the mitigation suggestion? : Database hardening should be applied
according to the least privilege principle. Built-in accounts should not be used, Only
required permissions should be granted. Connection should only be possible from
allowed hosts. (2 Points)
==========
● Vulnerability: Insecure Storage of Password (1 Point)
● Why does it arise? : On db_init.py function and MySQL table “users” it can clearly be
seen that passwords are stored in cleartext. (2 Points)
● What is the mitigation suggestion? : Passwords should be stored in a way that
prevents them from being obtained by an attacker even if the application or database
is compromised. Passwords should be hashed in order to slow down the brute force
process by selecting resource-intensive algorithms. Salting should be applied with
unique, randomly generated strings as part of the hashing process. (2 Points)
==========
● Vulnerability : Improper Error Handling (1 Points)
● Why does it arise? : At line 41, the application returns error details directly to the user
instead of logging details and returning generic error messages. (2 Points)
● What is the mitigation suggestion? : Application should be designed in a way that
handles all possible errors gracefully. The application should not reveal any detail
about the error and should log details. While logging, applications should avoid
logging any sensitive information as well. (2 Points)
Note: Applicants who only mentioned about lack of logging mechanism gained (3 points)
==========
Some of the applicants mentioned other vulnerabilities as well. (Some mentioned in the
notes part of answers) A couple of them need further functionalities added to the code. All of
these were evaluated separately and given 3 points each if they were identified correctly,
described sufficiently, and suggested correct mitigations. These topics can be found below.
● Lack Of Http Cookie Flags
● Insecure Transport (HTTP instead of HTTPS)
● Lack of Session Management, log-out mechanism, and token invalidation.
● XSS on line 101 in app.py
● Outdated Components in use. “pycrypto”

- Lack of Bruteforce Restriction on Login Form
- Lack of Multi-Factor Authentication