Week 1 task for a Cyber security internship!

Muhammad Saqib DHC-35

Task Breakdown:

---

1. Understanding Cyber security Fundamentals

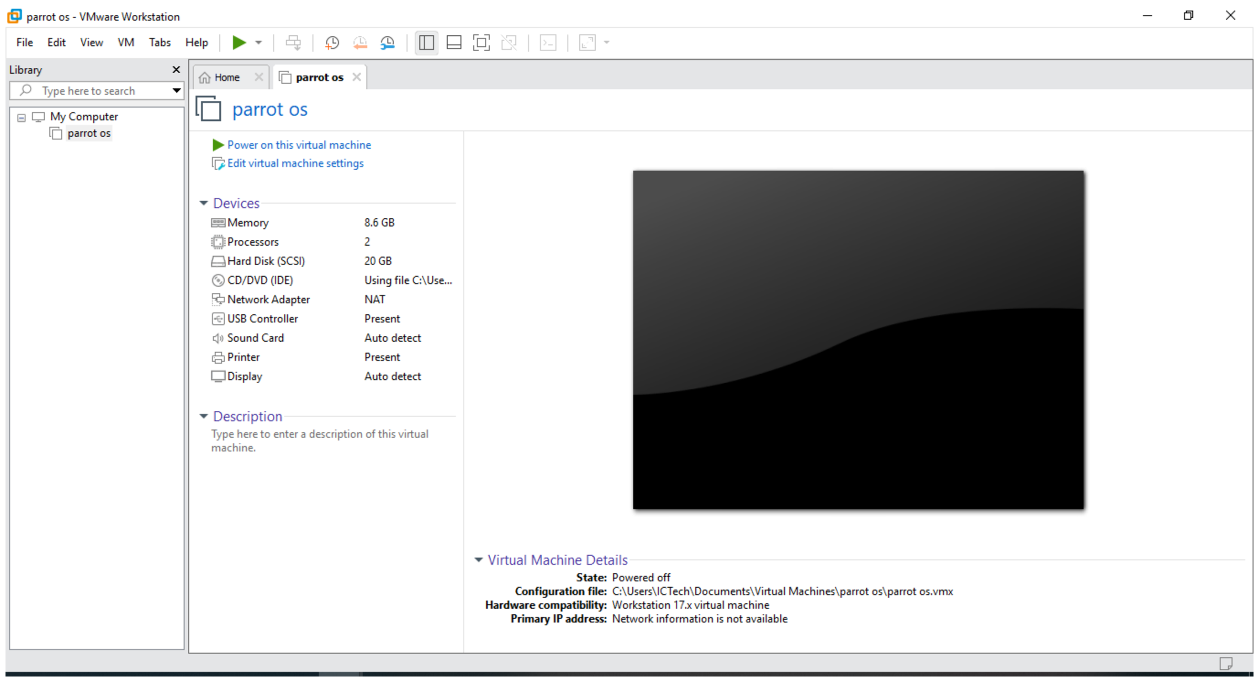
Objective: Research and summarize key cyber security concepts.

Key Concepts to Cover:

- **Confidentiality, Integrity, and Availability (CIA Triad):**
    - **Confidentiality** ensures that data is only accessible to those who have the right permissions.
    - **Integrity** ensures that data is accurate and not tampered with.
    - **Availability** ensures that data is accessible when needed.

- **Types of Cyber Attacks:**
    - **Phishing:** Deceptive attempts to obtain sensitive information (e.g., passwords) by masquerading as a trustworthy entity.
    - **Malware:** Malicious software that damages or disables computers (viruses, worms, trojans).
    - **Denial-of-Service (DoS):** Attacks that overwhelm a system, making it unavailable to its intended users.

- **Importance of Network Security, Data Protection, and User Authentication:**
    - **Network Security** focuses on protecting the network from threats.
    - **Data Protection** involves safeguarding sensitive data.
    - **User Authentication** verifies the identity of users to ensure they have the necessary access.


parrot os - VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
parrot os
Home
parrot os
Power on this virtual machine
Edit virtual machine settings
Devices
Description
Virtual Machine Details

parrot os

Memory	8.6 GB
Processors	2
Hard Disk (SCSI)	20 GB
CD/DVD (IDE)	Using file C:\Use...
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

Type here to enter a description of this virtual machine.

State: Powered off
Configuration file: C:\Users\ICTech\Documents\Virtual Machines\parrot os\parrot os.vmx
Hardware compatibility: Workstation 17.x virtual machine
Primary IP address: Network information is not available

<u>In details:</u>

Confidentiality, Integrity, and Availability (CIA Triad)

The **CIA Triad** is a foundational model in cybersecurity that describes the three key principles for protecting information systems:

1. **Confidentiality**:
   - **Definition**: Confidentiality ensures that only authorized individuals or systems can access sensitive information. It involves restricting access to data based on permissions and roles. This is vital in protecting sensitive data, such as personal, financial, and healthcare information.

   - **Methods**:
     - **Encryption**: Converting data into a format that can only be read by someone with the correct decryption key.
     - **Access Control**: Using user permissions, authentication systems (passwords, biometrics), and role-based access control (RBAC) to limit who can view or edit data.
     - **Data Masking**: Hiding sensitive data elements, often used in databases, so that unauthorized users cannot see or use it.

   - **Example**: If a company stores customer credit card information, they would use encryption to ensure that only authorized staff with the correct permissions can access the data.

2. **Integrity**:
   - **Definition**: Integrity ensures that data remains accurate, consistent, and unaltered throughout its lifecycle. This protects against unauthorized modification, corruption, or loss of data.

   - **Methods**:
     - **Hashing**: A cryptographic technique that converts data into a fixed-size string of characters, which can be used to verify the integrity of data. Any change in the data will result in a completely different hash.
     - **Checksums and CRCs**: These are mathematical algorithms that check the integrity of data by generating a small value based on the content of the data. If the data is altered, the checksum or CRC will not match.
     - **Digital Signatures**: A technique used to verify that data has not been tampered with during transmission.

   - **Example**: A banking application uses hashing to ensure that the amounts in user transaction records are not altered. If someone tried to modify the amount in a transaction, the system would detect the hash mismatch and alert administrators.

3. **Availability**:
   - **Definition**: Availability ensures that data and services are accessible and usable when needed by authorized users. This involves maintaining systems and infrastructure to avoid downtime and ensure reliability.

   - **Methods**:
     - **Redundancy**: Having multiple systems or backup mechanisms in place to ensure availability, such as backup servers, data replication, or cloud storage.
     - **Load Balancing**: Distributing network or system traffic across multiple servers to prevent any single server from becoming overwhelmed.
     - **Disaster Recovery Plans**: Having procedures in place to restore systems and data in case of a failure (e.g., backup data recovery).

        - **Uptime Monitoring**: Continuously monitoring the health of critical systems to detect issues before they cause service disruptions.

    - **Example**: A cloud service provider uses redundant data centers and load balancing to ensure that their services are available 24/7, even if one server or data center experiences a failure.

Types of Cyber Attacks

Cyberattacks are actions aimed at compromising the confidentiality, integrity, or availability of information or systems. Here are some common types:

1. **Phishing**:
    - **Definition**: Phishing is a social engineering attack in which attackers impersonate legitimate organizations or individuals to trick victims into revealing sensitive information like usernames, passwords, or credit card details.
    - **Methods**:
        - **Email Phishing**: Attackers send fraudulent emails that look like they're from a trusted source (e.g., a bank, email provider, or online retailer), urging the recipient to click on a malicious link or attachment.
        - **Spear Phishing**: A more targeted version of phishing, where attackers customize messages to specific individuals or companies based on information they've gathered.
        - **Vishing**: Voice phishing, where attackers use phone calls to impersonate legitimate entities and request sensitive information.
    - **Example**: A user receives an email that looks like it's from their bank, asking them to log in to verify their account information. If they click the link, they are taken to a fake website that steals their login credentials.
2. **Malware**:
    - **Definition**: Malware (malicious software) is a general term for any software designed to harm, exploit, or disrupt computer systems, networks, or devices.
    - **Types of Malware**:
        - **Viruses**: Programs that attach themselves to legitimate files and spread to other systems when those files are shared.
        - **Worms**: Self-replicating programs that spread across networks without the need for user intervention.
        - **Trojans**: Malware that disguises itself as legitimate software, tricking users into downloading it, at which point it can steal data, install other malware, or provide unauthorized access.
        - **Ransomware**: A type of malware that locks or encrypts a user's files and demands a ransom to release them.

    - **Example**: A computer becomes infected with a trojan, giving an attacker access to sensitive files and allowing them to steal personal information.
3. **Denial-of-Service (DoS)**:
    - **Definition**: A Denial-of-Service attack aims to overwhelm a system, network, or service, making it unavailable to users by flooding it with excessive requests.
    - **Types**:
        - **Flooding Attack**: The attacker floods a system with traffic (e.g., sending thousands of requests to a website) to overload it and cause it to crash.
        - **Distributed Denial-of-Service (DDoS)**: A more powerful version of DoS, where the attack is distributed across multiple machines, often infected with malware (botnets), to generate high volumes of traffic from multiple sources, making it harder to stop.
    - **Example**: An attacker uses a botnet to flood a website with requests, causing it to slow down or crash, making it inaccessible to legitimate users.

Importance of Network Security, Data Protection, and User Authentication

1. **Network Security**:
    - **Definition**: Network security refers to the practices, policies, and tools used to protect computer networks from threats, attacks, or unauthorized access.
    - **Importance**:
        - Network security safeguards against data breaches, malware, and other attacks targeting the network layer.
        - It ensures the secure communication of sensitive information, like financial data or personal information, over the network.
    - **Tools & Techniques**:
        - **Firewalls**: Used to filter network traffic and block unauthorized access.
        - **Intrusion Detection Systems (IDS)**: Monitor network traffic for suspicious activities or attacks.
        - **Virtual Private Networks (VPNs)**: Encrypt network traffic to ensure secure communication over the internet.
2. **Data Protection**:
    - **Definition**: Data protection involves safeguarding sensitive information from corruption, unauthorized access, and loss, while ensuring it is available and accurate when needed.
    - **Importance**:
        - Data protection is critical in industries like healthcare, finance, and government, where sensitive personal and business data is constantly handled.
        - Protecting data is necessary to maintain trust and comply with laws and regulations (e.g., GDPR, HIPAA).
    - **Methods**:
        - **Encryption**: Protects data by transforming it into unreadable formats.

        - **Backup & Recovery**: Regularly backing up data and having disaster recovery strategies ensures data is not lost.
        - **Data Masking and Redaction**: Techniques to obscure or remove sensitive data from non-production environments or public view.
3. **User Authentication**:
    - **Definition**: User authentication is the process of verifying the identity of a user, device, or system before granting access to resources.
    - **Importance**:
        - Authentication ensures that only authorized users can access sensitive data or systems.
        - Strong authentication methods help prevent unauthorized access, data breaches, and identity theft.
    - **Methods**:
        - **Password-based Authentication**: The most common method but requires strong, unique passwords.
        - **Multi-factor Authentication (MFA)**: Requires more than one form of verification, such as a password combined with a fingerprint or SMS code.
        - **Biometric Authentication**: Uses physical characteristics, such as fingerprints or facial recognition, to authenticate users.

Together, these concepts form the foundation of cybersecurity, guiding practices that help protect systems, networks, and data from various cyber threats. Implementing robust network security measures, protecting data, and ensuring strong user authentication are essential for maintaining trust and safeguarding against cyber attacks.

---

4. Password Security and Hashing

Password security is a critical component of protecting sensitive data and preventing unauthorized access. One of the most commonly used techniques for securing passwords is **hashing**. Additionally, adding a **salt** to a password before hashing further strengthens security by preventing certain types of attacks, such as **rainbow table** attacks. Let’s dive deeper into both concepts.

Hashing

What is Hashing?

Hashing is the process of converting a password (or any piece of data) into a fixed-length string of characters using a cryptographic function (called a **hash function**). A **hash function** takes an input (like a password) and produces a unique, fixed-length string (the hash). The output is always the same length, regardless of the size of the input data.

- **Important Characteristics of Hashing**:
    - **One-way function**: The process is irreversible, meaning you can't get the original password back from the hash.
    - **Deterministic**: The same input will always produce the same hash.
    - **Fast**: Hash functions are designed to be quick to compute.
    - **Unique**: A good hash function will produce unique outputs for different inputs. Even small changes to the input will result in drastically different hashes.

Why is Hashing Used?

The primary purpose of hashing is to securely store passwords. Instead of storing the actual password in a database, systems store the hash of the password. When users log in, the system hashes the password they enter and compares it to the stored hash. If the two hashes match, the password is correct, but the actual password is never stored.

Popular Hashing Algorithms:

- **SHA-256**: Part of the SHA-2 family, SHA-256 is widely used for security purposes because it provides a strong cryptographic hash.
- **MD5**: While once popular, MD5 is no longer recommended due to vulnerabilities that make it susceptible to collisions.
- **bcrypt** and **PBKDF2**: These algorithms are designed to be slower, making brute-force attacks more difficult.

Salting

What is Salting?

Salting is the practice of adding random data (a **salt**) to a password before it is hashed. The salt is typically unique for each user and is stored along with the hashed password. The primary purpose of salting is to prevent **rainbow table** attacks, which are precomputed tables of hashes for common passwords.

Why Add a Salt?

- **Rainbow Tables**: These are precomputed tables of hash values for commonly used passwords. Without a salt, attackers can simply look up the hash of a password in the

table to recover the original password. By adding a unique salt to each password, even identical passwords will have different hashes, making rainbow table attacks ineffective.
- **Brute Force Resistance**: Salting also ensures that attackers cannot easily guess multiple hashes by adding additional complexity to the hash.

How Does Salting Work?

A salt is a random string that is generated for each password. The salt is concatenated with the password, and then the resulting string is hashed. This makes the hashed password unique even if multiple users have the same password.

Python Script for Hashing and Salting

Here's how you can implement password hashing and salting in Python using the **`hashlib`** module, which provides a way to hash data using various algorithms (e.g., SHA-256).

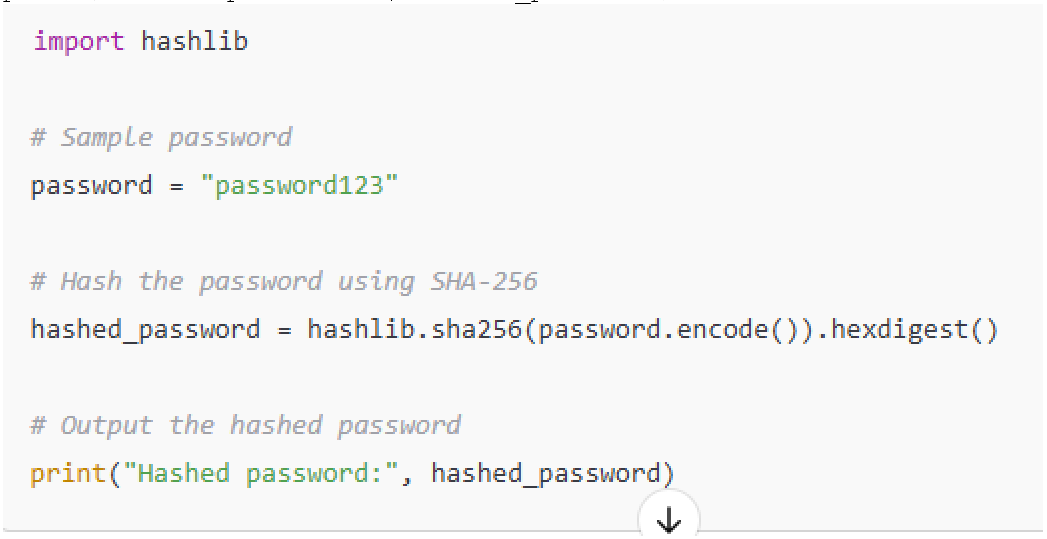
Basic Hashing Example (No Salt)

```
import hashlib

# Sample password
password = "password123"

# Hash the password using SHA-256
hashed_password = hashlib.sha256(password.encode()).hexdigest()

# Output the hashed password
print("Hashed password:", hashed_password)
```



```
import hashlib

# Sample password
password = "password123"

# Hash the password using SHA-256
hashed_password = hashlib.sha256(password.encode()).hexdigest()

# Output the hashed password
print("Hashed password:", hashed_password)
```

Explanation:

- `password.encode()` converts the password to bytes, which is required for hashing.
- `hashlib.sha256()` applies the SHA-256 hashing function to the password.
- `.hexdigest()` converts the hash to a readable hexadecimal format.

Salting and Hashing Example

To improve security, we can add a unique salt before hashing the password.

```
import hashlib
import os

# Sample password
password = "password123"

# Generate a unique salt (random string of 16 bytes)
salt = os.urandom(16).hex()

# Combine the password and salt
salted_password = password + salt

# Hash the salted password using SHA-256
salted_hashed_password = hashlib.sha256(salted_password.encode()).hexdigest()

# Output the salt and the salted hash
print("Salt:", salt)
print("Salted Hashed password:", salted_hashed_password)
```

```
import hashlib
import os

# Sample password
password = "password123"

# Generate a unique salt (random string of 16 bytes)
salt = os.urandom(16).hex()

# Combine the password and salt
salted_password = password + salt

# Hash the salted password using SHA-256
salted_hashed_password = hashlib.sha256(salted_password.encode()).hexdigest()

# Output the salt and the salted hash
print("Salt:", salt)
print("Salted Hashed password:", salted_hashed_password)
```

Explanation:

- `os.urandom(16).hex()` generates a 16-byte random salt and converts it to a hexadecimal string.
- The salt is concatenated with the password before hashing.
- The result is a unique salted hash for each password, even if multiple users have the same password.

Why Hashing and Salting are Essential

1. **Hashing**:
   - **Security**: It protects the actual password by not storing it directly in the system. Even if the password database is compromised, the attacker cannot directly obtain the plaintext passwords.
   - **Efficiency**: Hashing algorithms are designed to be fast and efficient, making them suitable for comparing passwords during login attempts.
2. **Salting**:
   - **Prevents Rainbow Table Attacks**: Without salting, attackers can use precomputed rainbow tables to easily reverse-engineer common passwords. Adding a unique salt to each password ensures that even identical passwords will have different hashes.
   - **Brute Force Resistance**: The addition of a salt increases the complexity of the hash, making it more resistant to brute-force attacks.

Example Use Case of Hashing and Salting in Real Life

Let’s consider an example where an attacker tries to hack into a website's password database. Without salting, the attacker could use a **rainbow table** to crack the hashes of common passwords. However, if each password is salted before hashing, the attacker would have to generate a new rainbow table for every single password, which would significantly slow down the attack.

For example:

- If **two users** have the same password (`password123`), the hashes will still be different if a unique salt is added for each user.
- Without salting, both users would have the same hash (`ef92b778bafe18fbb4d4b71ea6ebc5f7bb3643a3ffdd2f2acb7f5a456a4d4a07`), making it easy for the attacker to guess the password.
- With salting, each user would have a different salted hash even though they used the same password, making it more difficult for the attacker to crack them.

Deliverables

1. **Python Script**:
   - The Python script should include both the basic password hashing and salted password hashing examples.
   - Include the salt generation process (`os.urandom()` or a fixed salt) to demonstrate the importance of salting.
2. **Explanation**:
   - Provide a clear explanation of **why** hashing is used (to protect passwords) and **why salting** is important (to prevent rainbow table and brute-force attacks).
   - Discuss how hashing and salting enhance security in real-world applications.
3. **Hashcat** (optional):
   - If you are using a tool like **Hashcat** to perform the hashing, you would share screenshots showing the tool in action, though Python scripting is more common for demonstrating hashing in a basic cybersecurity context.

Conclusion

In cybersecurity, securing passwords through **hashing** and **salting** is a fundamental practice that helps protect user data from attackers. By understanding and implementing these concepts, you are taking crucial steps toward ensuring the security of systems and preventing common password-related vulnerabilities.

---

5. Basic Threat Identification

Objective: Identify common security threats and vulnerabilities.

Steps:

- **Research OWASP Top 10:**
  - Review the **OWASP Top 10** security risks, which include:

    1. **Injection Attacks** (e.g., SQL injection)
    2. **Broken Authentication**

3. **Sensitive Data Exposure**
4. **XML External Entities (XXE)**
5. **Broken Access Control**
6. **Security Misconfiguration**
7. **Cross-Site Scripting (XSS)**
8. **Insecure Deserialization**
9. **Using Components with Known Vulnerabilities**
10. **Insufficient Logging & Monitoring**

1. Injection Attacks (e.g., SQL Injection)

Description:

Injection attacks occur when an attacker sends malicious data to an interpreter (such as a database query processor) through a vulnerable input field. The attacker can inject their own code or commands that are executed by the system, allowing them to manipulate the application's behavior.

How It Is Exploited:

- **SQL Injection (SQLi)**: An attacker can insert malicious SQL code into an input field (like a login form or search box). If the application does not properly validate or sanitize the input, the injected SQL query can interact with the database, allowing the attacker to retrieve, modify, or delete sensitive data.

Example:

- Suppose a login page has a form where users enter their username and password. If the input fields are not properly sanitized, an attacker might enter the following into the username field:
- `' OR '1'='1`

  This alters the SQL query to always return true, potentially allowing unauthorized access to the system without needing a valid username or password.

---

2. Cross-Site Scripting (XSS)

Description:

Cross-Site Scripting (XSS) involves injecting malicious scripts into web pages that are viewed by other users. These scripts can steal session cookies, perform actions on behalf of the user, or redirect the user to malicious websites.

How It Is Exploited:

- **Stored XSS**: The attacker injects a malicious script into an application (like a comment or profile page). When other users visit the page, the script executes in their browser.
- **Reflected XSS**: The attacker sends a malicious URL to a victim. If the victim clicks the link, the script is executed in their browser, potentially stealing their session or redirecting them to a malicious site.

Example:

- An attacker might post a comment on a website with the following malicious script:
- `<script>alert('Your session cookie is: ' + document.cookie);</script>`

  When another user views the comment, the script executes in their browser and could steal their session cookie, allowing the attacker to impersonate them.

---

3. Cross-Site Request Forgery (CSRF)

Description:

Cross-Site Request Forgery (CSRF) tricks a user into performing actions they did not intend to. The attacker exploits the trust a website has in the user's browser by making an unauthorized request on behalf of the victim, often without their knowledge.

How It Is Exploited:

- If a user is logged into a website (e.g., a banking application) and an attacker sends them a specially crafted link or form, the attacker can force the user to perform actions like transferring money, changing account details, or deleting files, without the user’s consent.

Example:

- Suppose a user is logged into their bank account and visits a malicious website that contains the following HTML code:

-
  ```
  <img src="https://bank.com/transfer?amount=1000&to=attackerAccount"
  style="display:none;">
  ```

  When the user visits the website, the image request triggers a fund transfer to the attacker’s account without the user’s knowledge, as the bank system trusts the logged-in user’s credentials.

---

4. Broken Authentication and Session Management

Description:

Broken authentication and session management vulnerabilities occur when web applications fail to properly manage user authentication or session tokens. This allows attackers to hijack sessions or bypass authentication mechanisms.

How It Is Exploited:

- If an application doesn't securely handle session tokens (e.g., using weak or predictable session IDs), attackers can easily guess or steal session tokens to impersonate legitimate users.
- Poorly implemented logout functionality, failure to expire sessions after a period of inactivity, or failure to use secure cookies can also leave sessions vulnerable to hijacking.

Example:

- An attacker might steal a session ID by intercepting network traffic (via a man-in-the-middle attack) and use it to impersonate the user. This can be done if the application doesn't use secure transmission (like HTTPS) or the session ID is not protected (e.g., no HttpOnly or Secure flags).

---

5. Security Misconfiguration

Description:

Security misconfiguration refers to improper or incomplete configurations in web applications, servers, or databases that expose vulnerabilities. These misconfigurations can include default passwords, unnecessary services running, or overly permissive access controls.

How It Is Exploited:

- Attackers can exploit default settings, such as default administrator passwords or unsecured services that are left active. They can also take advantage of incorrect permissions or open directories that are exposed to the public.

Example:

- A web server might have an unsecured management interface that is accessible from the internet. If the server has not been configured to restrict access to certain IPs or require strong authentication, an attacker could access the interface and gain control over the server.

Summary of Threats

Threat	Description	Example of Exploitation
SQL Injection	Malicious SQL commands injected into input fields to manipulate databases.	Attacker inserts SQL code to bypass login or retrieve sensitive data.
Cross-Site Scripting (XSS)	Malicious scripts injected into web pages viewed by other users, potentially stealing session data.	Malicious script steals cookies or performs actions on behalf of a user.
Cross-Site Request Forgery (CSRF)	Attacker tricks a logged-in user into performing an unintended action.	User unknowingly transfers money by visiting a malicious website.
Broken Authentication and Session Management	Weak session management or improper authentication handling.	Attacker steals or hijacks session tokens to impersonate a user.
Security Misconfiguration	Improper configuration of application, server, or database security settings.	Default passwords or unsecured services allow unauthorized access.

Conclusion

Recognizing and mitigating these common web application vulnerabilities is essential in maintaining a secure environment. Each of these threats can be exploited in different ways, but they can all be prevented or mitigated by following best practices such as input validation, proper session management, and configuring security controls appropriately. Understanding these threats, as described in the OWASP Top 10, is a vital step in building secure web applications.