Task 1
The Open Web Application Security Project (OWASP) Top 10 is a critical framework that identifies the ten most significant security risks to web applications. Updated biennially, the 2021 edition addresses the evolving threat landscape, providing a comprehensive overview of vulnerabilities that developers and security professionals must prioritize. Each entry in the Top 10 highlights a specific risk, along with examples and mitigation strategies, making it an essential resource for building secure software. This report documents the tasks completed on TryHackMe related to the OWASP Top 10 (2021), offering practical insights into each vulnerability and emphasizing the importance of proactive security measures.

Task 2
Objective: Set up OpenVPN to connect securely to the target machine for OWASP Top 10 tasks.

Steps:

1. **Install OpenVPN**: Download and install OpenVPN from the official website.
2. **Download Configuration**: Obtain the OpenVPN configuration file (.ovpn) from the TryHackMe challenge page.
3. **Import and Connect**: Import the configuration file into the OpenVPN client and connect to the VPN using your TryHackMe credentials.
4. **Verify Connection**: Check your IP address to confirm the VPN connection is active.
5. **Access Target Machine**: Use the specified IP address to access the target machine and complete the tasks.

Task 3
Broken Access Control

Description: Broken access control refers to a security vulnerability that occurs when an application does not properly restrict users' access to resources or functions. This allows unauthorized users to perform actions or access data they shouldn't be able to.

Common Issues:

- Lack of role-based access controls
- Insecure direct object references
- Missing authorization checks on APIs

Impact: Attackers can exploit broken access controls to gain unauthorized access to sensitive data, perform administrative actions, or escalate privileges, leading to data breaches or unauthorized transactions.

Mitigation Strategies:

- Implement strict access controls based on user roles.
- Ensure that all sensitive actions are properly authorized.

Task 4
Challenge Name: Insecure Direct Object Reference (IDOR)
Category: Web Application Security
Objective: Exploit an IDOR vulnerability to access unauthorized resources within a web application.

Overview:

The IDOR challenge demonstrates the risks of insecure direct object references, where attackers can manipulate URLs or parameters to access restricted resources.

Steps to Complete:

1. **Access the Challenge**: Open the challenge on TryHackMe and review the instructions.
2. **Inspect the Application**: Use browser developer tools to analyze requests, focusing on URL parameters.
3. **Identify Vulnerability**: Look for patterns in the URL that suggest an IDOR issue (e.g., user IDs).
4. **Manipulate Parameters**: Change parameter values to access other users' data or restricted actions.
5. **Capture the Flag**: Successfully exploit the vulnerability to retrieve the flag.
6. **Document Findings**: Record the steps taken and the flag for submission.

Task 5

Cryptographic Failures

Description: Cryptographic failures occur when weak cryptographic practices or insufficient protection of sensitive data lead to vulnerabilities. This can include improper implementation of cryptographic algorithms, insecure key management, or using outdated protocols.

Common Issues:

- Use of weak or outdated encryption algorithms (e.g., MD5, SHA-1).
- Inadequate key management practices, such as hardcoding keys in source code.
- Failure to use secure protocols (e.g., not enforcing HTTPS).

Impact: Attackers can exploit cryptographic failures to gain access to sensitive data, such as passwords or personal information, leading to data breaches and unauthorized access.

Mitigation Strategies:

- Implement strong, industry-standard cryptographic algorithms and protocols.
- Regularly review and update cryptographic practices to align with current best practices.
- Use secure key management solutions to protect encryption keys.

Task 6
Flat-file databases are single files used to store large amounts of data, making them easily accessible from various locations. Unlike traditional databases, which often require dedicated servers like MySQL or MariaDB, flat-file databases simplify setup and are suitable for smaller web applications.

However, if a flat-file database is stored within the web server's root directory, it becomes accessible to users visiting the website. This can lead to **Sensitive Data Exposure**, as users may download and query the database, gaining unauthorized access to sensitive information.

The most common format for flat-file databases is SQLite, which is widely supported and can be queried using the `sqlite3` command-line client. This allows for easy interaction with the database across different programming languages and platforms.

Task 7
In the previous task, we learned how to query an SQLite database to find sensitive data, including user password hashes. This task focuses on cracking those hashes.

While Kali Linux includes various tools for hash cracking, this material will utilize the online tool Crackstation, which is effective for cracking weak password hashes. The hashes encountered in this challenge are primarily weak MD5 hashes, making them suitable for Crackstation's capabilities. For more complex hashes, more advanced tools would be necessary.

Task 8
Challenge Name: Cryptographic Failure
Category: Web Application Security
Objective: Identify and exploit cryptographic weaknesses in a web application.

Overview:

The Cryptographic Failure challenge focuses on vulnerabilities related to weak cryptographic practices, particularly poor password hashing and insecure data storage.

Steps to Complete:

1. **Access the Challenge**: Open the challenge on TryHackMe and review the instructions.
2. **Examine the Application**: Analyze the application for user authentication features and sensitive data storage.
3. **Identify Weak Hashing Algorithms**: Look for the use of weak hashing algorithms, such as MD5, for password storage.
4. **Extract Password Hashes**: Obtain any exposed password hashes from the application.
5. **Crack Password Hashes**: Use an online tool like Crackstation to crack the weak password hashes.
6. **Document Findings**: Record the steps taken and any cracked passwords.
7. **Capture the Flag**: Retrieve the flag once access is gained through the cracked credentials.

Task 9

Injection

Description: Injection vulnerabilities occur when an attacker can send untrusted data to an interpreter as part of a command or query. This can lead to the execution of unintended commands or access to sensitive data.

Common Types:

- **SQL Injection**: Manipulating SQL queries to access or modify data in the database.
- **Command Injection**: Executing arbitrary commands on the host operating system via a vulnerable application.
- **LDAP Injection**: Modifying LDAP queries to gain unauthorized access to directory services.

Impact: Successful injection attacks can lead to data breaches, loss of data integrity, and unauthorized access to system resources.

Mitigation Strategies:

- Use prepared statements and parameterized queries to avoid direct insertion of user input into commands.
- Validate and sanitize user inputs to prevent malicious data from being processed.
- Employ web application firewalls (WAFs) to detect and block injection attempts.

Task 10

Injection Commands Summary

1. SQL Injection:

- **Description**: Involves manipulating SQL queries by injecting malicious SQL code.
- **Common Commands**:
  - `OR '1'='1'` - Bypasses authentication.
  - `UNION SELECT` - Retrieves data from other tables.

2. Command Injection:

- **Description**: Allows attackers to execute arbitrary commands on the server.
- **Common Commands**:
  - `; ls` - Executes a command to list files.
  - `&&` or `||` - Chains commands to execute additional commands conditionally.

3. LDAP Injection:

- **Description**: Involves manipulating LDAP queries to access unauthorized data.
- **Common Commands**:
  - `*)(uid=*)` - Bypasses user authentication filters.
  - `|` - Combines multiple search filters.

4. XML Injection:

- **Description**: Manipulates XML data to alter the intended structure.
- **Common Commands**:
  - `<user><name>attacker</name></user>` - Inserts malicious XML content.
  - `<!--` - Comments out sections of the XML.

5. XSS (Cross-Site Scripting):

- **Description**: Injects malicious scripts into web pages viewed by other users.
- **Common Commands**:
  - `<script>alert('XSS')</script>` - Executes a script when injected into a page.
  - `javascript:` - Executes JavaScript in the context of the page.

Task 11

Challenge Focus: Insecure Design
Objective: Identify vulnerabilities related to poor design in a web application.

Overview:

Insecure design vulnerabilities stem from fundamental flaws in the application's architecture rather than issues in the implementation. These vulnerabilities can allow attackers to exploit assumptions made during the design phase, such as the inability of a user to utilize numerous IP addresses for brute-forcing a numeric code. Fixing these issues often requires significant redesign and redevelopment efforts.

Practical Example:

- **Task**: Access Joseph's account via the application at `http://MACHINE_IP:85`.
- **Identified Weakness**: The password reset mechanism contains a design flaw that can be exploited. Users can potentially reset passwords without adequate verification, allowing unauthorized access.

Recommendation:

To prevent such vulnerabilities, it's crucial to conduct threat modeling during the early stages of the development lifecycle. Implementing a Secure Software Development Lifecycle (SSDLC) can help identify and mitigate these risks early on.

Task 12

Overview: Security misconfigurations occur when security settings are not properly configured, leaving applications vulnerable despite using the latest software versions. This type of vulnerability can result from various factors, including poor permissions, default accounts with unchanged passwords, and the presence of debugging interfaces in production environments.

Common Examples of Security Misconfigurations:

- Poorly configured cloud service permissions (e.g., S3 buckets).
- Unnecessary features or services enabled.
- Default accounts and passwords not changed.
- Overly detailed error messages exposing system information.
- Absence of HTTP security headers.

Debugging Interfaces:

One notable instance of security misconfiguration is the exposure of debugging features in production software. For example, the Werkzeug console, part of Python-based web applications, provides a debugging interface that can execute arbitrary Python code. If this console is left accessible in a production environment, attackers can exploit it to run malicious commands.

Case Study: In 2015, Patreon experienced a security breach allegedly due to an open Werkzeug debug interface. A security researcher had previously reported the vulnerability, highlighting the risk associated with leaving debugging tools enabled in production.

Conclusion:

To prevent security misconfigurations, developers should ensure that debugging features are disabled before deployment and conduct regular security assessments to identify and rectify misconfigured settings.

Task 13

Overview: Vulnerable components refer to outdated or unpatched libraries, frameworks, or software modules that may contain known security vulnerabilities. These components can pose significant risks when integrated into applications, potentially allowing attackers to exploit them.

Key Points:

- **Types of Vulnerable Components**: Common vulnerable components include third-party libraries, frameworks, and software dependencies that have not been updated or patched.
- **Impact**: Exploiting vulnerabilities in these components can lead to data breaches, unauthorized access, and other severe security incidents.

- **Common Issues**:
    - Using unsupported or end-of-life software versions.
    - Failing to monitor for security updates and patches.
    - Ignoring known vulnerabilities in publicly available databases (e.g., CVE).

Mitigation Strategies:

- **Regular Updates**: Keep all components updated to the latest versions, applying security patches as they become available.
- **Dependency Management**: Utilize tools that can automatically check for vulnerable components and alert developers to necessary updates.
- **Security Testing**: Incorporate vulnerability scanning and testing as part of the development lifecycle to identify and remediate vulnerable components proactively.

Task 14

Overview: Remote Code Execution (RCE) vulnerabilities allow attackers to execute arbitrary code on a remote system. Understanding how to identify and exploit these vulnerabilities is crucial for penetration testers.

Key Points:

- **Script Documentation**: Most exploit scripts provide clear documentation on the required arguments, minimizing the need to sift through extensive code to understand usage.
- **Version Identification**: While some exploits may offer straightforward version numbers, others may require examining HTML source or making educated guesses to identify vulnerabilities. Familiarity with known vulnerabilities can facilitate this process.
- **Research Importance**: Effective exploitation often hinges on conducting thorough research to determine the specific version of the application and its associated vulnerabilities.

Conclusion:

RCE vulnerabilities exemplify a critical aspect of the OWASP Top 10, where prior research and knowledge significantly aid penetration testers in identifying and exploiting weaknesses. This emphasizes the importance of ongoing research and understanding of known vulnerabilities in the security landscape.

Task 15

Lab Name: Vulnerable Component
Objective: Identify and exploit vulnerabilities in a web application at `http://MACHINE_IP:84`.

Overview:

In this lab, participants assess a vulnerable application to identify outdated or insecure components that could be exploited.

Steps Taken:

1. **Access the Application**:
    - Navigated to `http://MACHINE_IP:84` and reviewed the application.
2. **Identify Vulnerable Components**:
    - Inspected the application for outdated libraries and frameworks, checking for version numbers and using scanning tools to identify known vulnerabilities.
3. **Research Vulnerabilities**:
    - Utilized online resources, such as the CVE database, to find vulnerabilities related to the identified components.
4. **Exploit the Vulnerability**:
    - Used available exploit scripts or crafted custom payloads to take advantage of the vulnerabilities.
5. **Document Findings**:
    - Recorded all steps, including vulnerabilities identified, methods of exploitation, and any sensitive data retrieved.

Conclusion:

The lab highlighted the critical need for keeping software components updated and securely configured. Vulnerable components pose significant security risks, making awareness and proactive management essential.

Task 16

Key Points:

1. **Weak Password Policies**:
    - Use of easily guessable or common passwords increases the risk of unauthorized access. Applications should enforce strong password complexity and length requirements.
2. **Poor Session Management**:
    - Insecure session handling, such as failing to expire sessions after logout or using predictable session identifiers, can allow attackers to hijack user sessions.
3. **Credential Storage Issues**:
    - Storing passwords in plaintext or using weak hashing algorithms makes it easier for attackers to compromise user credentials.
4. **Account Lockout Mechanisms**:
    - Lack of account lockout policies can lead to brute-force attacks, allowing attackers to repeatedly attempt to guess passwords without consequence.
5. **Multi-Factor Authentication (MFA)**:

    - Not implementing MFA weakens authentication security. MFA adds an additional layer of verification, making it harder for attackers to gain unauthorized access.
6. **Failure to Implement Proper Authentication**:
    - Applications may not properly verify user identities, allowing unauthorized users to gain access to sensitive functions or data.

Task 17

Lab Name: Identification and Authentication Failure
Objective: Identify vulnerabilities in authentication mechanisms within a web application.

Steps to Complete the Lab:

1. **Access the Application**:
    - Navigate to the designated TryHackMe lab URL.
2. **Examine the Login Page**:
    - Review security features like password requirements and account lockout policies.
3. **Test Weak Passwords**:
    - Attempt to log in using common passwords to identify any vulnerabilities.
4. **Brute Force Attack Simulation**:
    - Use a tool (e.g., Hydra) to test the application's resilience against brute force attacks.
5. **Check Credential Storage**:
    - Inspect how passwords are stored (plaintext vs. hashed) and look for exposed credentials.
6. **Assess Session Management**:
    - Check for session expiration policies and active sessions post-logout.
7. **Explore Multi-Factor Authentication (MFA)**:
    - Determine if MFA is implemented and attempt to bypass it.
8. **Document Findings**:
    - Record identified vulnerabilities and methods of exploitation.
9. **Recommendations**:
    - Suggest improvements for password policies, MFA implementation, and session management.

Task 18

Overview: Software and data integrity failures occur when applications fail to protect their software and data from unauthorized modifications. These vulnerabilities can lead to the execution of malicious code or unauthorized access to sensitive information.

Key Points:

1. **Inadequate Integrity Checks**:

    - Applications may lack proper validation mechanisms to ensure that software and data have not been altered. This can allow attackers to introduce malicious code or manipulate data.
2. **Use of Unsigned Code**:
    - Relying on unsigned or unverified code can expose applications to risks. Attackers may exploit this by substituting legitimate components with compromised versions.
3. **Dependency Vulnerabilities**:
    - Applications often rely on third-party libraries or components. If these dependencies are not regularly updated and verified for integrity, they can introduce vulnerabilities.
4. **Insufficient Logging**:
    - Failing to log and monitor changes to software and data can hinder the detection of unauthorized modifications and make it challenging to respond to incidents.
5. **Attacks on Update Mechanisms**:
    - Attackers may target software update mechanisms to deliver malicious updates or tamper with the update process, compromising the integrity of the application.

Task 19

Overview: Web applications often rely on third-party libraries, such as jQuery, loaded from external servers. While convenient, this practice introduces significant security risks.

Risks:

1. **Dependency on External Sources**:
    - If the external server goes down, your website may break.
2. **Malicious Modifications**:
    - Compromised servers can lead to injected malicious code.
3. **Version Control Issues**:
    - Automatic updates may expose your app to vulnerabilities in newer library versions.
4. **Lack of Integrity Checks**:
    - Without checks, there's no assurance the library is unaltered.
5. **Legal and Compliance Risks**:
    - Potential licensing issues with third-party libraries.

Task 20

Overview: A data integrity failure was found in some libraries implementing JSON Web Tokens (JWTs), allowing attackers to bypass signature validation and gain unauthorized access.

Vulnerability Details:

Exploit Method:

1. **Modify the Header**: Change the `alg` field to `"none"`.
2. **Remove the Signature**: Delete the signature portion while keeping the dot.

Original -
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImV4YW1wbGUiLCJleHBpcmF0aW9uIjoxNjA0MjQ3NjIwfQ.S5xP8Pvg1vLZyZPz_y8gOdIb_3B2w6kzNqJ3MvgtPRM

Manipulated -
eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJ1c2VybmFtZSI6ImFkbWluIiwiaWF0IjoxNjA0MjQ3NjIwfQ.

Implications:

Attackers can gain admin access without valid credentials, leading to potential data breaches.

Task 21

Logging user actions in web applications is essential for tracing attacker activities during security incidents. Without logging, risks include regulatory penalties for failing to protect personally identifiable information (PII) and the potential for undetected further attacks.

Key Log Information:

- HTTP status codes
- Time stamps
- Usernames
- API endpoints/page locations
- IP addresses

Logs must be securely stored in multiple locations due to their sensitivity.

Monitoring Importance:

Proactive monitoring helps detect suspicious activities, such as unauthorized login attempts, unusual IP addresses, automated tool usage, and known attack payloads.

Response to Suspicious Activity:

It's vital to assess suspicious actions by impact level, prioritizing higher-impact events to trigger alerts for prompt response.

Task 22

SSRF Vulnerability

Summary: Server-Side Request Forgery (SSRF) allows attackers to make unauthorized requests from a server, potentially leading to serious consequences such as network enumeration, exploiting trust relationships, and remote code execution (RCE).

Potential SSRF Use Cases:

1. **Network Enumeration:** Identify internal IP addresses and ports.
2. **Exploiting Trust Relationships:** Access restricted services that trust the server.
3. **Interacting with Non-HTTP Services:** Communicate with non-HTTP services, potentially leading to RCE.

Steps to Access the Admin Area:

1. **Investigate the Application:** Identify input fields that may allow SSRF manipulation.
2. **Craft SSRF Payloads:** Use specific URLs to target internal services.
3. **Gain Access:** Follow any instructions to access the admin area.

Reported by Srija Paul

srijapaul078@gmail.com