WEEK-2

Asad Muhammd Channar

DHC-56

Task-4:

Introduction to Web Vulnerabilities (OWASP Top 10)

Required:

- Choose one vulnerability from the OWASP Top 10 (e.g., SQL Injection or XSS).
- Research a realworld example of the vulnerability being exploited and write a 1page summary of the attack, including how it could have been prevented.

OWASP TOP 10

The **OWASP Top 10** is a widely recognized list that highlights the most critical security vulnerabilities affecting web applications. Created by the Open Web Application Security Project (OWASP), this list serves as a valuable resource for developers, security professionals, and organizations aiming to improve their application security posture. Updated regularly, the OWASP Top 10 addresses the evolving landscape of cyber threats, providing insights into common attack vectors and security weaknesses. Each vulnerability is accompanied by detailed explanations, real-world examples, and guidance on mitigation strategies. By focusing on these top vulnerabilities, organizations can prioritize their security efforts, allocate resources effectively, and educate their teams about potential risks. Understanding the OWASP Top 10 is essential for building secure applications and protecting sensitive data from malicious actors.

SQL Injection:

SQL Injection (SQLi) is one of the most dangerous vulnerabilities in web applications and is ranked high in the OWASP Top 10. It occurs when an attacker can manipulate a website's SQL database by injecting malicious SQL queries through user input fields, such as search bars, login forms, or URL parameters. This type of attack can lead to unauthorized access to sensitive information, data theft, or even full control over the database.

Equifax Data Breach

One of the most infamous examples of SQL Injection exploitation is the **Equifax data breach** in 2017, which exposed sensitive personal information of over 147 million people, including Social Security numbers, birth dates, addresses, and driver's license numbers. This breach is a textbook case of SQL Injection that caused massive damage, both financially and reputationally.

The Attack

In the Equifax data breach, the attackers exploited a vulnerability in a web application known as **Apache Struts**, which Equifax used to develop and run their public-facing web applications. Apache Struts is an open-source web application framework that allows developers to build dynamic websites.

In March 2017, a vulnerability (CVE-2017-5638) was discovered in the Struts framework, which allowed attackers to perform **Remote Code Execution (RCE)** through specially crafted requests. However, Equifax failed to apply the necessary security patches despite the public availability of the fix.

The attackers used this vulnerability to send malicious requests to Equifax's servers, allowing them to inject malicious SQL queries. These queries compromised Equifax's SQL database, allowing the attackers to exfiltrate sensitive data over the course of several months. The breach wasn't discovered until July 2017, after which Equifax disclosed the massive data breach to the public.

Impact of the Attack

How It Could Have Been Prevented

The Equifax breach was a result of poor security practices, and the following measures could have prevented the attack:

1. **Timely Application of Security Patches**: The vulnerability in Apache Struts was publicly known, and a security patch was available months before the breach occurred. Had Equifax promptly applied the patch, the attackers would not have been able to exploit the SQL Injection vulnerability. Keeping software and web frameworks up-to-date is critical in preventing such attacks.

2. **Input Validation and Parameterized Queries**: Implementing **input validation** and using **parameterized queries** or **prepared statements** can prevent SQL Injection attacks. These methods ensure that user input is treated as data and not executable code, thus protecting against malicious SQL queries. By enforcing strict validation of user inputs, the Equifax application would have rejected the malformed requests used in the attack.
3. **Web Application Firewalls (WAFs)**: A **Web Application Firewall** could have detected and blocked malicious requests before they reached the database. WAFs are designed to filter out common web-based attacks, including SQL Injection attempts, by analyzing incoming HTTP requests and rejecting suspicious traffic patterns.
4. **Regular Security Audits and Penetration Testing**: Equifax failed to properly secure its systems through routine security audits. Regular **penetration testing** could have identified the vulnerable application before the attackers exploited it. By simulating real-world attacks, Equifax could have uncovered vulnerabilities in their systems and mitigated them before they were compromised.
5. **Segmentation of Sensitive Data**: Equifax did not properly segment its network, allowing attackers to access highly sensitive personal data directly once they breached the system. Proper network segmentation could have restricted the attacker's access to critical data, limiting the overall damage.