Week 1 Assignment

Name: Shaz Alam **reg.no: 21BCY10221**

Owasp Top 10 vulnerabilities:

1. Injection

Injection occurs when an attacker exploits insecure code to insert (or inject) their own code into a program. Because the program is unable to determine code inserted in this way from its own code, attackers are able to use injection attacks to access secure areas and confidential information as though they are trusted users. Examples of injections include SQL injections, command injections, CRLF injections, and LDAP injections.

Application security testing can reveal injection flaws and suggest remediation techniques such as stripping special characters from user input or writing parameterized SQL queries.

2. Broken Authentication

Incorrectly implemented authentication and session management calls can be a huge security risk. If attackers notice these vulnerabilities, they may be able to easily assume legitimate users' identities.

Multifactor authentication is one way to mitigate broken authentication. Implement DAST and SCA scans to detect and remove issues with implementation errors before the code is deployed.

3. Sensitive Data Exposure

APIs, which allow developers to connect their applications to third-party services like Google Maps, are great time-savers. However, some APIs rely on insecure data transmission methods, which attackers can exploit to gain access to usernames, passwords, and other sensitive information.

Data encryption, tokenization, proper key management, and disabling response caching can all help reduce the risk of sensitive data exposure.

4. XML External Entities

This risk occurs when attackers are able to upload or include hostile XML content due to insecure code, integrations, or dependencies. An SCA scan can find risks in third-party components with known vulnerabilities and will warn you about them. Disabling XML external entity processing also reduces the likelihood of an XML entity attack.

5. Broken Access Control

If authentication and access restriction are not properly implemented, it's easy for attackers to take whatever they want. With broken access control flaws, unauthenticated or unauthorized users may have access to sensitive files and systems, or even user privilege settings.

Configuration errors and insecure access control practices are hard to detect as automated processes cannot always test for them. Penetration testing can detect missing authentication, but other methods must be used to determine configuration problems. Weak access controls and issues with credentials management are preventable with secure coding practices, as well as preventative measures like locking down administrative accounts and controls and using multi-factor authentication.

6. Security Misconfiguration

Just like misconfigured access controls, more general security configuration errors are huge risks that give attackers quick, easy access to sensitive data and site areas.

Dynamic testing can help you discover misconfigured security in your application.

7. Cross-Site Scripting

With cross-site scripting, attackers take advantage of APIs and DOM manipulation to retrieve data from or send commands to your application. Cross-site scripting widens the attack surface for threat actors, enabling them to hijack user accounts, access browser histories, spread Trojans and worms, control browsers remotely, and more.

Training developers in best practices such as data encoding and input validation reduces the likelihood of this risk. Sanitize your data by validating that it’s the content you expect for that particular field, and by encoding it for the “endpoint” as an extra layer of protection.

8. Insecure Deserialization

Deserialization, or retrieving data and objects that have been written to disks or otherwise saved, can be used to remotely execute code in your application or as a door to further attacks. The format that an object is serialized into is either structured or binary text through common serialization systems like JSON and XML. This flaw occurs when an attacker uses untrusted data to manipulate an application, initiate a denial of service (DoS) attack, or execute unpredictable code to change the behavior of the application.

Although deserialization is difficult to exploit, penetration testing or the use of application security tools can reduce the risk further. Additionally, do not accept serialized objects from untrusted sources and do not use methods that only allow primitive data types.

9. Using Components with Known Vulnerabilities

No matter how secure your own code is, attackers can exploit APIs, dependencies and other third-party components if they are not themselves secure.

A static analysis accompanied by a software composition analysis can locate and help neutralize insecure components in your application. Veracode's static code analysis tools can help developers find such insecure components in their code before they publish an application.

10. Insufficient Logging and Monitoring

Failing to log errors or attacks and poor monitoring practices can introduce a human element to security risks. Threat actors count on a lack of monitoring and slower remediation times so that they can carry out their attacks before you have time to notice or react.

To prevent issues with insufficient logging and monitoring, make sure that all login failures, access control failures, and server-side input validation failures are logged with context so that you can identify suspicious activity.

Penetration testing is a great way to find areas of your application with insufficient logging too. Establishing effective monitoring practices is also essential.

.