Web Application Security

Web Application Security

- Web AppSec is the idea of building websites to function as expected, even when they are under attack.
- The concept involves a collection of security controls engineered into a Web application to protect its assets from potentially malicious agents
- Web applications, like all software, inevitably contain defects

Web Application Security

- Some of these defects constitute actual vulnerabilities that can be exploited, introducing risks to organizations
- Web application security defends against such defects
- It involves
  - leveraging secure development practices and
  - implementing security measures throughout SDLC, ensuring that design-level flaws and implementation-level bugs are addressed

Importance of Web Security Testing

- To find security vulnerabilities in Web applications and their configuration

- The primary target is the application layer (i.e., what is running on the HTTP protocol)

- Involves sending different types of input to provoke errors and make the system behave in unexpected ways

- These so called “negative tests” examine whether the system is doing something it isn’t designed to do.

Importance of Web Security Testing

- To understand that Web security testing is not only about testing the security features (e.g., authentication and authorization) that may be implemented in the application
- It is equally important to test that other features are implemented in a secure way (e.g., business logic and the use of proper input validation and output encoding)
- The goal is to ensure that the functions exposed in the Web applications are secure.

Types of Web Security Testing

1. Dynamic Application Security Test (DAST)
2. Static Application Security Test (SAST)
3. Penetration Test
4. Runtime Application Self Protection (RASP)

Dynamic Application Security Test (DAST)

- This automated application security test is best for internally facing, low-risk applications that must comply with regulatory security assessments
- For medium-risk applications and critical applications undergoing minor changes,
- combining DAST with some manual web security testing for common vulnerabilities is the best solution

Static Application Security Test (SAST)

- This application security approach offers automated and manual testing techniques

- It is best for identifying bugs without the need to execute applications in a production environment
-
- It also enables developers
  - to scan source code and
  - systematically find and eliminate software security vulnerabilities.

Penetration Test

- This manual application security test is best for critical applications, especially those undergoing major changes

- The assessment involves business logic and adversary-based testing to discover advanced attack scenarios

Runtime Application Self Protection (RASP)

- This evolving application security approach
  - encompasses a number of technological techniques to instrument an application
  - so that attacks can be monitored as they execute and, ideally, blocked in real time

Runtime Application Self Protection (RASP)

- This evolving application security approach
    - encompasses a number of technological techniques to instrument an application
    - so that attacks can be monitored as they execute and, ideally, blocked in real time

How does application security testing reduce your organization’s risk?

- **Majority of Web Application Attacks**
- SQL Injection
- XSS (Cross Site Scripting)
- Remote Command Execution
- Path Traversal

How does application security testing reduce your organization’s risk?

- **Attack Results**
- Access to restricted content
- Compromised user accounts
- Installation of malicious code
- Lost sales revenue
- Loss of trust with customers
- Damaged brand reputation
- And much more

How does application security testing reduce your organization’s risk?

- The several of the top attacks used by attackers, which can result in serious damage to an individual application or the overall organization

- Knowing the different attacks that make an application vulnerable,

- in addition to the potential outcomes of an attack, allow your firm to preemptively address the vulnerabilities and accurately test for them.

How does application security testing reduce your organization’s risk?

- By identifying the root cause of the vulnerabilities, mitigating controls can be implemented during the early stages of the SDLC to prevent any issues

- Additionally, knowledge of how these attacks work can be leveraged to target known points of interest during a Web application security test.

- Recognizing the impact of an attack is also key to managing your firm’s risk, as the effects of a successful attack can be used to gauge the vulnerability’s total severity

How does application security testing reduce your organization's risk?

- If issues are identified during a security test, defining their severity allows your firm to efficiently prioritize the remediation efforts.

- Start with critical severity issues and work towards lower impact issues to minimize risk to your firm

What features should be reviewed during a web application security test?

- The following non-exhaustive list of features should be reviewed during Web application security testing
- An inappropriate implementation of each could result in vulnerabilities, creating serious risk for your organization

What features should be reviewed during a web application security test?

- **Application and server configuration**
- Potential defects are related to
  - encryption/cryptographic configurations,
  - Web server configurations, etc.

- **Input validation and error handling**
- SQL injection, cross-site scripting (XSS), and
- other common injection vulnerabilities are the result of poor input and output handling.

What features should be reviewed during a web application security test?

- **Authentication and session management**
  - Vulnerabilities potentially resulting in user impersonation
  - Credential strength and protection should also be considered

- **Authorization**
- Testing the ability of the application to protect against vertical and horizontal privilege escalations.

What features should be reviewed during a web application security test?

- **Business logic**
- These are important to most applications that provide business functionality

- **Client-side logic**
- With modern, JavaScript-heavy web pages, in addition to web pages using other types of client-side technologies (e.g., Silverlight, Flash, Javaapplets), this type of feature is becoming more prevalent

Web Application Security Threats

- Each year, attackers develop inventive web application security threats
  - to compromise sensitive data and
  - access their targets' database

- Consequently, security experts build on the exploited vulnerabilities and strengthen their systems through their learning's every year

Injection Attacks

- A web app that is vulnerable to injection attacks accepts untrusted data from an input field without any proper sanitation

- By typing code into an input field, the attacker can trick the server into interpreting it as a system command and thereby act as the attacker intended

- Some common injection attacks include SQL injections, Cross-Site Scripting, Email Header Injection, etc.

- These attacks could lead to unauthorized access to databases and exploitation of admin privileges.

Injection Attacks

- **How to prevent:**
- Keep untrusted inputs away from commands and queries
- Use a safe Application Programming Interface (API) that avoids interpreters or uses parameterized interfaces
- Filter and sanitize all inputs as per a white list.
- This prevents the use of malicious character combinations.

Broken Authentication

- Broken authentication is an umbrella term given to vulnerabilities wherein authentication and session management tokens are inadequately implemented

- This improper implementation allows hackers
  - to make claims over a legitimate user’s identity,
  - access their sensitive data, and
  - potentially exploit the designated ID privileges.

Broken Authentication

- **How to prevent:**
- End sessions after a certain period of inactivity.
- Invalidate a session ID as soon as the session ends.
- Place limiters on the simplicity of passwords.
- Implement multi-factor authentication (2FA/MFA).

Cross Site Scripting (XSS)

- It is an injection-based client-side attack
- This attack involves injecting malicious code in a website application to execute them in the victims' browsers
- Any application that doesn't validate untrusted data adequately is vulnerable to such attacks
- Successful implementation results in
  - theft of user session IDs,
  - website defacing, and
  - redirection to malicious sites (thereby allowing phishing attacks).

Cross Site Scripting (XSS)

- **How to prevent:**
- Encode all user-supplied data.
- Use auto-sanitization libraries such as OWASP’s AntiSamy.
- White list inputs to disallow certain special character combinations.

Insecure Direct Object References (IDOR)

- Mostly through manipulation of the URL, an attacker gains access to database items belonging to other users

- For instance, the reference to a database object is exposed in the URL.

- The vulnerability exists when someone can edit the URL to access other similar critical information (such as monthly salary slips) without additional authorization.

Insecure Direct Object References (IDOR)

- **How to prevent:**
- Implement proper user authorization checks at relevant stages of users’ web app journey.

- Customize error messages so that they don’t reveal critical information about the respective user.

- Try not to disclose reference to objects in the URL; use POST based information transmission over GET.

Security Misconfigurations

- According to OWASP top 10, this is the most common web application security threats found across web applications
-
- This vulnerability exists because developers and administrators “forget” to change some default settings such as default passwords, usernames, reference IDs, error messages, etc.
- Given how easy it is to detect and exploit default settings that were initially placed to accommodate a simple user experience, the implications of such a vulnerability can be vast once the website is live: from admin privileges to complete database access.

Security Misconfigurations

- **How to prevent:**
- Frequently maintain and update all web application components: firewalls, operating systems, servers, databases, extensions, etc.
- Make sure to change default configurations.
- Make time for regular penetration tests (though this applies to every vulnerability that a web app could have).

Unvalidated Redirects and Forwards

- Pretty much every website redirects a user to other web pages.
- When the credibility of this redirection is not assessed, the website leaves itself vulnerable to such URL based attacks.
- A malicious actor can redirect users to phishing sites or sites containing malware.
- Phishers search for this vulnerability extensively since it makes it easier for them to gain user trust.

Unvalidated Redirects and Forwards

- **How to prevent:**
- Avoid redirection where possible.
- Give the destination parameters a mapping value rather than the actual URL
- Let the server-side code translate the mapping value to the actual URL.

Missing Function Level Access Control

- mostly similar to IDOR
- The core differentiating factor between the two is that IDOR tends to give the attacker access to information in the database
- In contrast, Missing_ Function Level Access Control _allows the attacker access to special functions and features that should not be available to any typical user
- Like, IDOR, access to these functions can be gained through URL manipulation as well

Missing Function Level Access Control

- **How to prevent:**
- Implement adequate authorization measures at relevant stages of user web app use.

- Deny all access to set features and functions unless attempted by a pre-approved (admin) user.

- Allow for a flexible shift in grant and rejection of access to feature privileges in your code. Hence, allowing a practical and secure shift in privilege access when needed.