WEB APPLICATIONS

Web applications themselves can be enticing targets: A web app's vulnerability can be used by an attacker to steal user data, carry out fraud, infect users with malware, and much more.

Web apps can also act as a very noticeable entry point for an attacker trying to get further into the network of an organization: An attacker might target the servers the application runs on instead of the application itself, installing a backdoor to get direct access to them before switching to other computers on the network.

Web applications are sought after because they are visible, easily accessible, and provide a wide range of possible rewards for attackers. Due to the popularity of web applications as targets, groups like the Open Web Application Security Project (OWASP) have been established to inform people about protecting web apps. The OWASP Top Ten is a list of the 10 most serious categories of web application vulnerabilities that OWASP releases.

Most of the threats described in this article can be somewhat avoided by adhering to one straightforward rule: Never trust user input.

Injection Attacks

For some of its functionality, many web apps rely on backend software; some of this infrastructure, such as databases, receives queries based on user input. An injection attack occurs when a malicious query is made by the attacker to deceive the software into performing an action that it shouldn't.

LDAP Injection

A network directory can be accessed via the LDAP protocol. An injection attack against this protocol is called LDAP Injection. Since LDAP frequently deals with credentials like usernames and passwords, a malicious LDAP query can have a significant security impact. Some web applications generate LDAP queries from user input.

SQL Injection

SQL Injection is an injection attack that utilizes SQL, a language used to query databases. Malicious uses for SQL injection attacks include accessing or altering data as well as sending commands to the OS that runs the database. Web applications frequently employ user input to do database queries; a prime example of this is the search function.

Preventing Injection Attacks

Make sure that user input cannot be utilized to influence the software that responds to a query if a web application will be making a query based on user input. While input sanitization, which verifies that queries don't contain control characters used to fool software, is one approach to accomplish this, it is preferable to organize queries in ways that are intrinsically resistant to injection, such as SQL's parameterized queries.

XSS And CSRF

Cross-Site Scripting (XSS) and Cross-Site Request Forgery target the frontend of an online application, whereas injection attacks target the backend.

Directory Traversal

A website's structure may be comparable to the organization of a filesystem on the server it is hosted on. When an attacker has access to restricted areas of the filesystem, this is known as directory traversal, which in file paths means "the folder above this one," is typically used to do this. The simplest method is to just include the.. es in the URL, as in www.example.com/../../...

This can be used to read files that really shouldn't be read, such as the /etc/passwd or /etc/shadow files on Linux, which store user information and password hashes respectively.

Cross-Site Request Forgery

When you are currently authenticated on one website, cross-site request phishing allows that website to send requests to another website. Since this is a little unclear, consider it like this:

Your bank account is open in one tab with you logged in. 2 You open a second tab and browse a malicious website.

The malicious website contains HTML that instructs you to ask your bank's website to do a malicious act.

Your browser delivers the request in accordance with the HTML code.

Your browser sends the request to your bank's website. It fulfills the request's wicked wishes because, as far as it is aware, it is from you.

CSRF tokens, which are randomly generated values created for each session and must be supplied in requests for them to be executed, can protect against this type of vulnerability, which frequently makes use of cookies saved in the browser.

Cross-Site Scripting

When user input on a website is read as JavaScript and executed, this is known as cross-site scripting. XSS vulnerabilities can range in severity from low to extremely high based on the type of XSS and the function of the website.

Reflected XSS is when a website returns the malicious user input immediately, such as a search function displaying your query at the top of the screen. You would need to deceive other people into doing anything, like clicking a URL that had the malicious query included as a parameter, to influence them. Although still potentially harmful, this is not nearly as risky as...

Stored XSS refers to situations where a website stores user input, such as in a forum or post on a social media platform. The input may be significantly more hazardous because it can be seen by many individuals. This kind of XSS has been utilized for everything from sizable-scale credit card data theft to relatively innocent practical jokes like self-sharing social media posts.

Conclusion

Attackers frequently target web applications, both as standalone targets and as entry points to internal networks. When creating online applications, designers should take vulnerabilities like XSS, CSRF, Directory Traversal, and injection attacks into account. By not trusting user input, several of these vulnerabilities can be mitigated to some extent. This entails validating input, ensuring control characters are handled as text and note code appropriately, and triple-checking the origin of requests.