



































**PREVENTIVE MEASURES**

**BRUTE FORCE:**

Limit failed login attempts

Make the root user inaccessible via SSH by editing the sshd\_config file

Don’t use a default port, edit the port line in your sshd\_configfile

Use Captcha

Limit logins to a specified IP address or range

Two factor authentication

**ALL XSS ATTACKS:**

Filter input on arrival: At the point where user input is received, filter as strictly as possible based on what is expected or valid input.

Encode data on output: At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.

Use appropriate response headers: To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.

Content Security Policy: As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.

Filter your inputs with a whitelist of allowed characters and use type hints or type casting. Escape your outputs with htmlentities and ENT\_QUOTES for HTML contexts, or JavaScript Unicode escapes for JavaScript contexts.

**CSRF ATTACKS:**

Assess the risk:

CSRF vulnerabilities do not apply to public content. They are only dangerous when authentication is required. Therefore, you can ignore this risk if you only have public content on your website. However, if you have a web application with user accounts, be extra vigilant. Treat CSRF as a major risk if you have an e-commerce application.

Use anti-CSRF tokens:

Anti-CSRF tokens are considered the most effective method of protecting against CSRF. Use a tested implementation such as CSRFGuard for Java or CSRFProtector for PHP to implement your anti-CSRF tokens. Develop your own mechanism only if there is no existing one for your environment.

Use SameSite cookies:

Set the SameSite attribute of your cookies to Strict. If this would break your web application functionality, set the SameSite attribute to Lax but never to None. Not all browsers support SameSite cookies yet, but most do. Use this attribute as additional protection along with anti-CSRF tokens.

**SQL INJECTION & BLIND SQL INJECTION ATTACKS:**

1) Continuous Scanning and Penetration Testing

The automated web application scanner has been the best choice to point out vulnerabilities within the web applications for quite some time now. Now, with SQL injections getting smarter in exploiting logical flaws, website security professionals should explore manual testing with the help of a security vendor.

They can authenticate user inputs against a set of rules for syntax, type, and length. It helps to audit application vulnerabilities discreetly so that you can patch the code before hackers exploit it to their advantage.

2) Restrict Privileges

It is more of a database management function, but enforcing specific privileges to specific accounts helps prevent blind SQL injection attacks. Begin with no privileges account and move on to ‘read-only’, ‘edit’, ‘delete’ and similar privilege levels.

Minimizing privileges to the application will ensure that the attacker, who gets into the database through the application, cannot make unauthorized use of specific data.

3) Use Query Parameters

Dynamic queries create a lot of troubles for security professionals. They have to deal with variable vulnerabilities in each application, which only gets graver with updates and changes. It is recommended that you prepare parameterized queries.

These queries are simple, easy to write, and only pass when each parameter in SQL code is clearly defined. This way, your info is supplied with weapons to differentiate between code and information inputs.

4) Instant Protection

A majority of organizations fail the problems like outdated code, scarcity of resources to test and make changes, no knowledge of application security, and frequent updates in the application. For these, web application protection is the best solution.

A managed web application firewall can be deployed for immediate mitigation of such attacks. It contains custom policies to block any suspicious input and deny information breach instantly. This way, you do not have to manually look for loopholes and mend problems afterward.