

# 1. Grunnprinsipp: kvifor treng vi sikring?

Når du lagar PHP-side som kommuniserer med ein database, er det mange moglege angrep:

- **SQL-injection** – når brukaren legg inn kode i eit felt for å manipulere databasen.
- **XSS (Cross-Site Scripting)** – når skadeleg JavaScript blir lagt inn i nettsida.
- **CSRF (Cross-Site Request Forgery)** – når ein annan nettside lurer brukaren til å utføre handlingar.
- **Dårleg passordhandtering** – når passord blir lagra i klartekst.

Målet vårt: hindre slike angrep med gode kodeteknikkar.

---

## 2. Bruk *prepared statements* for å hindre SQL-injection

**Feil og farleg måte:**

```
// ⚠️ Sårbar for SQL-injection!
$brukernavn = $_POST["brukernavn"];
$sql = "SELECT * FROM brukar WHERE brukernavn = '$brukernavn'";
$resultat = $conn->query($sql);
```

Ein brukar kan då skrive inn noko slikt:

```
admin' OR '1'='1
```

og få tilgang til alt.

---

**Trygg måte med prepared statements:**

```
// Kople til databasen
$conn = new mysqli("localhost", "root", "passord", "database");

// Sjekk tilkopling
if ($conn->connect_error) {
    die("Feil ved tilkopling: " . $conn->connect_error);
}

// Sikra spørring
$stmt = $conn->prepare("SELECT * FROM brukar WHERE brukernavn = ?");
$stmt->bind_param("s", $_POST["brukernavn"]); // "s" = string
$stmt->execute();

$resultat = $stmt->get_result();
while ($rad = $resultat->fetch_assoc()) {
    echo htmlspecialchars($rad["brukernavn"]);
}
```

 **Fordel:** Brukarens inndata blir aldri sett direkte inn i SQL-streng, så det kan ikkje køyre farleg kode.

---

## 3. Rens og kontroller all inndata

All data frå brukar (skjema, URL, cookie, API) må reknaast som *mistenkeleg*.

```
$navn = trim($_POST["navn"]); // Fjern mellomrom  
$navn = htmlspecialchars($navn); // Hindrar XSS ved vising
```

### **Eksempel – trygg lagring:**

```
if (!empty($_POST["tekst"])) {  
    $tekst = strip_tags($_POST["tekst"]); // Fjern HTML-taggar  
    $stmt = $conn->prepare("INSERT INTO kommentar (tekst) VALUES (?)");  
    $stmt->bind_param("s", $tekst);  
    $stmt->execute();  
}
```

---

## **4. Bruk `password_hash()` og `password_verify()` for passord**

### **Feil og farleg:**

```
$passord = $_POST["passord"];  
// ⚠️ Ikkje lagre i klartekst  
mysql_query($conn, "INSERT INTO brukar (brukernamn, passord) VALUES ('$brukarnamn', '$passord')");
```

### **Trygg måte:**

```
$hash = password_hash($_POST["passord"], PASSWORD_DEFAULT);  
$stmt = $conn->prepare("INSERT INTO brukar (brukernamn, passord) VALUES (?, ?)");  
$stmt->bind_param("ss", $_POST["brukarnamn"], $hash);  
$stmt->execute();
```

Når brukaren loggar inn:

```
$stmt = $conn->prepare("SELECT passord FROM brukar WHERE brukernamn = ?");  
$stmt->bind_param("s", $_POST["brukarnamn"]);  
$stmt->execute();  
$resultat = $stmt->get_result()->fetch_assoc();  
  
if (password_verify($_POST["passord"], $resultat["passord"])) {  
    echo "Innlogging vellukka!";  
} else {  
    echo "Feil passord.";  
}
```

---

## **5. Sikre vising mot XSS (Cross-Site Scripting)**

Når du viser tekst frå databasen:

```
// Feil - kan køyre JavaScript i nettlesaren  
echo $rad["kommentar"];
```

### Trygg måte:

```
echo htmlspecialchars($rad["kommentar"]);
```

Dette gjer at teikn som < og > blir vist som tekst, ikkje HTML.

---



## 6. Bruk session på trygg måte

```
session_start();  
session_regenerate_id(true); // Hindre session hijacking  
  
$_SESSION["brukar"] = $brukarnamn;  
  
// Når brukaren loggar ut:  
session_unset();  
session_destroy();
```

---



## 7. Bruk minst mogleg rettar i databasen

Lag ein eigen brukar for web-appen i MySQL:

```
CREATE USER 'webbrukar'@'localhost' IDENTIFIED BY 'sterktpassord';  
GRANT SELECT, INSERT, UPDATE, DELETE ON database.* TO 'webbrukar'@'localhost';
```

Ikkje bruk root for vanleg bruk!

## Teori: Kva er ei session?

- Ei *session* (økt) gjer at nettstaden kan **hugse informasjon mellom ulike sider eller lastingar**.
- Når `session_start();` køyrer, får brukaren ein unik ID (vanlegvis lagra som ein cookie).
- Alle variablar du lagrar i `$_SESSION` blir verande til du avsluttar økta eller slettar dei.

### Typiske bruksområde:

- innlogging og tilgangskontroll
  - handlekorg i nettbutikk
  - hugse val (språk, søker, tema osv.)
- 

### Praktisk eksempel: Søkjefeltet som hugsar førre søk

#### Forklaring av koden i praksis:

1. Brukaren skriv eit ord i søkerfeltet og trykkjer "Søk".
2. PHP lagrar søkerordet i `$_SESSION["forrige_soek"]`.
3. Neste gong sida lastar, vert dette søkerordet henta ut og vist i feltet.  
→ Brukaren slepp å skrive det inn på nytt.

Dette gjer opplevinga meir **personleg og brukarvenleg** – akkurat som når Google hugsar det du søker etter sist.

---

### Tryggleik og god praksis

#### Viktige poeng:

- `session_start();` må stå **før** all HTML-kode.
- `$_SESSION`-data blir lagra på serveren, ikkje i nettlesaren.
- Kombiner `session` med `htmlspecialchars()` for å unngå XSS når du viser data.
- I større prosjekt må ein hugse å **avslutte session** når brukaren loggar ut (`session_destroy();`).