

Kapittel 11

Sikkerhet og hacking

Læringsmål:

Sikkerhet er veldig viktig i websammenheng. Etter å ha jobbet med dette kapitlet skal du

- se behovet for å tenke sikkerhet i selv de enkleste løsninger
- forstå hvordan noen går frem for å ødelegge eller snike til seg informasjon de egentlig ikke skal få tilgang til
- kjenne til preventive mottiltak som kan gjøres for å unngå hacking
- vite om tiltak for å begrense adgang og foreta tilgangskontroll
- kunne validere data på tjenersiden
- forstå hvorfor sikker koding alene ikke er nok

11.1 Introduksjon til datasikkerhet

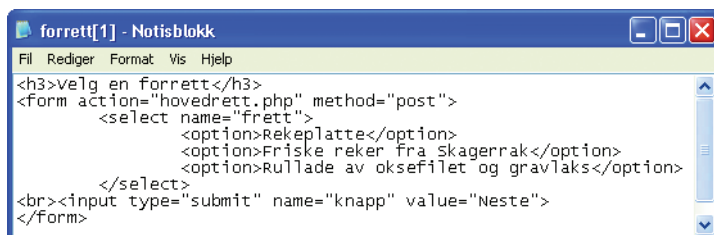
De fleste forbinder hackere med personer som utfører kriminelle handlinger i digitale sammenhenger. Strengt tatt trenger ikke hacking å innebære noe ulovlig, men siden begrepet for de fleste er såpass negativ ladet, benevnes ondsinnede handlinger som *hacking* i denne boka. Med hacking mener vi altså uønsket aktivitet av ymse slag.

I en verden hvor digitale ressurser og sensitive data skal være tilgjengelig fra hvor som helst, er sikkerhet essensielt. Vi skal i dette kapittelet sette fokus på potensielle farer og vise noen teknikker for å utnytte svakheter i vanlige systemer. Målet er ikke å utdanne hackere, men å utdanne programutviklere som kan bekjempe hacking ved å produsere robuste nettløsninger. Ikke prøv eksemplene som gjennomgås på eksisterende nettsted, for det kan få rettslige følger. En bedrift som har et dårlig sikret datasystem, kan fort anmelde deg om du forsøker å utføre «uskyldige» hack. Da nytter det lite å påstå at «det er jo dere som har laget et dårlig system». Det du betrakter som lovlig eller uskyldig, kan bli sett annerledes på av andre.

11.1.1 En restaurant som tilbyr nugatti i stedet for reker på loffen

Gjennom bruk av nedtrekkslister, radioknapper og liknende blir grensesnittet ikke bare mer intuitivt, men også en annen gevinst oppnås: Brukerens valgmuligheter begrenses slik at forventede data kan oversendes til videre behandling.

La oss nå anta at restauranten som ble skissert i avsnitt 7.2.4, er publisert under adressen <http://www.kokken.no> og ellers er uendret. Kunden komponerer sin meny ved å velge forret, hovedrett og dessert fra nedtrekkslistene. Bestillingen sendes deretter til kokken per e-post slik at han kan ha maten ferdig til kunden kommer. Bruk av nedtrekkslister (elementet `<SELECT>`) gjør at en bare kan gjøre ett valg. Stefan ble for en tid siden sparket grunnet dårlig hygiene og enkle menysammensetninger. Han vil nå spille den nye norske kokken Nikolai et puss, og starter med å se på kildekode for første side av bestillingen. Kildekode som sendes til klienten (i HTML), kan leses ved å velge riktig funksjonalitet i nettleseren. Koden til PHP-scriptet forlater aldri tjeneren og er derfor ikke tilgjengelig for brukeren.



```
forrett[1] - Notisblokk
Fil Rediger Format Vis Hjelp
<h3>velg en forret</h3>
<form action="hovedrett.php" method="post">
  <select name="frett">
    <option>Rekeplatte</option>
    <option>Friske reker fra Skagerrak</option>
    <option>Rullade av oksefilet og gravlaks</option>
  </select>
  <br><input type="submit" name="knapp" value="Neste">
</form>
```

Figur 11.1 HTML-kildekode kan avsløre mye for hackere.

Enhver besøkende kan se på kildekoden og dermed få frem en rekke opplysninger.

- Det er scriptet *hovedrett.php* som skal prosessere dataene videre.
- Data oversendes ved hjelp av POST-metoden.
- Feltet med valgt informasjon heter `frett`.

Dersom Stefan prøver å skrive følgende URL i nettleseren

<http://www.kokken.no/hovedrett.php?frett=tullematSomIkkeEksisterer>

vil han få problemer. Det betyr at `$_POST['frett']` brukes til å hente frem verdien som velges på forrett-siden. Siden informasjonen som skrives inn i spørrestrengen, legges i `$_GET['frett']`, og scriptet tydeligvis bruker `$_POST['frett']`, taper Stefan første runde.

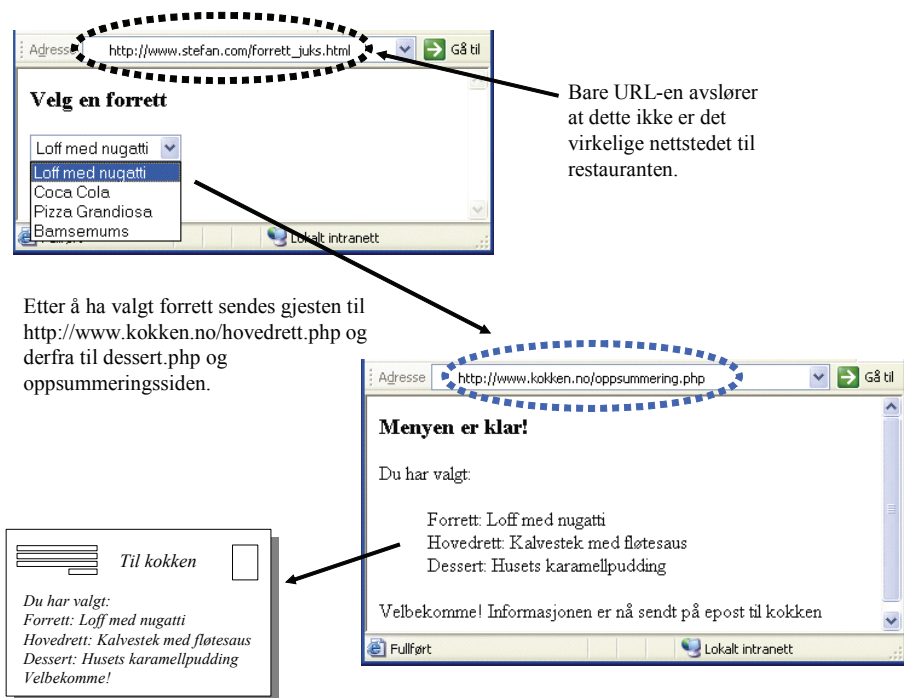
Men Stefan er lur. Han kopierer kildekoden, fikser de to XHTML-feilene i samme sleng (uten at det har noe med saken å gjøre), og legger ut en variant av koden som en ny side på sitt eget nettsted, nemlig <http://www.stefan.com>. Koden ser slik ut:

Kodesnutt 11.1 Et script på hackerens hjemmeside

```
<h3>Velg en forret</h3>
<form action="http://www.kokken.no/hovedrett.php"
      method="post">
  <select name="frett">
    <option>Loff med nugatti</option>
    <option>Coca Cola</option>
    <option>Pizza Grandiosa</option>
    <option>Bamsemums</option>
  </select>
  <br /><input type="submit" name="knapp" value="Neste" />
</form>
```

Legg merke til endringene i `<option>`-elementene. De som nå måtte besøke denne varianten på Stefan sitt nettsted, vil kunne velge helt andre ting enn det kultiverte utvalget kokken har satt opp. De vil også kunne tro at de er kommet på riktig side (for eksempel kan de bli lurt dit med en e-post). Forretten vil inneholde søpeldata, mens videre valg av hovedrett og dessert skjer på <http://www.kokken.no>.

Det ville også vært mulig å la brukeren velge alle rettene på det falske nettstedet og bare la oppsummeringen foretas på riktig nettsted. Resultatet er at informasjonen som vises på skjerm og sendes til kokken per e-post, er noe helt annet enn forventet:



Figur 11.2 En hacker kan lett lure andre til å sende ugyldige menyer til kokken.

11.1.2 Vårt fokus på sikkerhet så langt

For å kunne forstå behovet for sikkerhetstiltak er det nødvendig å innta en holdning om at brukere *ikke* alltid er snille og oppfører seg som forventet. Eksempelet med kokken som ble sint, bør sette i gang noen tanker knyttet til skadepotensialet. Vi har allerede satt noe fokus på sikkerhet, men det har kanskje druknet i de mange temaer, teknikker og muligheter som stadig har blitt introdusert. En liten oppsummering kan være nyttig.

I kapittel 1 ble samspillet mellom Apache og PHP forklart. Begge programmene styres av konfigurasjonsfiler som kan endres, for eksempel gjennom menyer som pakkeløsninger å la WAMP og MAMP tilbyr. Ved å la `display_errors=on` vil feilmeldinger vises, noe som er nyttig i utviklingsfasen, men skummelt etter publisering.

I kapittel 6 ble noen funksjoner som kan brukes i sikkerhetssammenheng, introdusert. Disse ble brukt videre i andre kapitler. Funksjonen `md5()` produserer en hash-sum, og nytten av dette skal vi se mer på senere i dette kapittelet.

Såkalt *Cross Site Scripting* (XSS) er en aktuell problematikk blant annet i forbindelse med gjestebøker (avsnitt 8.3.3). Dersom en hacker for eksempel lagrer et skummelt JavaScript i stedet for en virkelig hilsen, kan andre besøkende som skal lese gjesteboka, få problemer. Funksjonen `htmlentities()` er en måte å fore-

bygge angrep av denne art. En kanskje enda større fare utgjøres i forbindelse med opplasting av filer. Her kan en dårlig designet løsning gi hackeren mulighet til å laste opp sine egne PHP-script og kjøre disse. Denne formen for XSS-angrep er beskrevet i avsnitt 8.4.3.

Tilstandsbevaring har en rekke sikkerhetsmessige aspekter knyttet til seg, og spesielt viktig er det å sikre at data ikke lekker ut til uvedkommende. Målet med sessions er å ta vare på informasjon på tvers av forespørsler. Lagring skjer på tjeneren, men det er også nødvendig å overføre en ID mellom hver forespørsel for å kunne knytte riktig informasjonssett opp mot riktig klient.

Graden av sikkerhet med sessions avhenger av hvorvidt hver ID kan skjules for uvedkommende. Overføring av URL øker mulighetene for at en hacker kan foreta en vellykket session hijacking – stjeling av en sessionID og tilhørende data – hvorpå hackeren for eksempel vil kunne overta offerets gyldige innlogging. I forbindelse med nettbanker kan skaden bli enorm, men disse har heldigvis ofte andre mekanismer som hindrer slikt. Denne problematikken ble diskutert i avsnitt 7.4.4.

Tematikken knyttet til databaser og sikkerhet er så viktig at det får en større porsjon plass i dette kapitlet.

Mange bruker ferdiglagde PHP-løsninger av ulike slag, for eksempel phpMyAdmin, SquirrelMail og liknende. Slike er lettvinne i bruk, og en kan også lære av hvordan flinke *PHP-programmerere* har bygd opp slike komplekse løsninger. Siden PHP ikke kompiles, er koden selvsagt fritt tilgjengelig. Dette betyr også at hackere kan analysere koden og finne svakheter, for så å bruke kunnskapene i angrep mot nettsteder som gjør bruk av slike systemer. Det er i så måte viktig å holde programvaren oppdatert og lese dokumentasjonen godt.

Det ble også nevnt i kapittel 7 at protokollen *SSL (Secure Sockets Layer)* bør brukes i forbindelse med sensitive opplysninger, for eksempel i en nettbank. SSL krypterer informasjon som sendes over HTTP. Du har sikkert sett URL-er som starter med `https://` – dette betyr at SSL er aktivert. Det er derimot ikke nok bare å skrive inn `https://` i stedet for `http://` for å bruke SSL. Først må SSL-støtte være installert på tjeneren på den siden du skal besøke. Deretter må et nøkkelpar og et sertifikat skaffes slik at tjeneren kan identifisere seg overfor de besøkende. Hvem som helst kan sertifisere seg selv og lage egne sertifikater. Seriøse nettsteder som skal bruke SSL, bør bruke et sertifikat som er signert av en betrodd tredjepart (for eksempel Thawte eller Verisign), og en slik prosess koster penger.

Det fine er at de som skriver PHP-kode, ikke må ta stilling til om SSL er i bruk. Krypteringen med SSL foregår på et lavere nivå enn det programkjøringen befinner seg på. Funksjoner i PHP fungerer uavhengig av om overføringen av informasjon mellom klienten og tjeneren skjer kryptert. For en grundig gjennomgang av denne lagdelingen, anbefales boka *Innføring i datakommunikasjon* av Øyvind Hallsteinen, Bjørn Klefstad og Olav Skundberg.

11.2 Sikkerhet ved bruk av databaser

Hacking kan gi langt større konsekvenser enn ugyldige menyer som sendes per e-post. Nesten alle større nettsteder har et databasesystem i bunn. Mange nettsteder har et åpent område som er eksponert for hele verden, og et internt område hvor bare spesielt utvalgte har tilgang. Bedrifter, høgschooler, nettbutikker, portaler, stat og kommune og så videre er eksempler på slike.

Behovet for å tilby fleksibilitet til de privilegerte brukerne gjør at de fleste tyr til en slik dobbel løsning over Internett. En bedrift kan slik tilby sine ansatte tilgang til nyttige interne dokumenter fra hele verden (også hjemmefra) uten at de må være fysisk tilkoblet bedriftens interne nettverk.

11.2.1 Riktig brukernavn og passord må til for å logge inn, eller ...?

Vi oppretter nå en tabell som kan lagre brukere og passord, med følgende enkle oppbygning:

```
CREATE TABLE brukere (
    id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    brukernavn VARCHAR(10),
    passord VARCHAR(20)
);
```

Nøkkelfeltet er ikke viktig, men tas med for ordens skyld. Brukernavn og tilhørende passord vil ligge i samme rad. Dette er vist i oversikten under. Slike tabeller med brukernavn og passord er vanlige i mange systemer som har innloggingsfunksjonalitet, og et PHP-script må ta seg av innloggingen og foreta passordsjekk opp mot databasetabellen.

Tabell med brukernavn og passord

```
SELECT * FROM brukere
```

id	brukernavn	passord
1	per	1234
2	line	shhhj
3	sah	hemmelig
4	karinor	bamse
5	ola	ola65

Strategien for passordsjekk er enkel: Dersom antall rader er større enn eller lik 1, må det bety at både passordet og brukernavnet ble funnet i tabellen minst én gang og *i samme rad*. Dermed må det være riktig bruker som logger seg inn. I motsatt

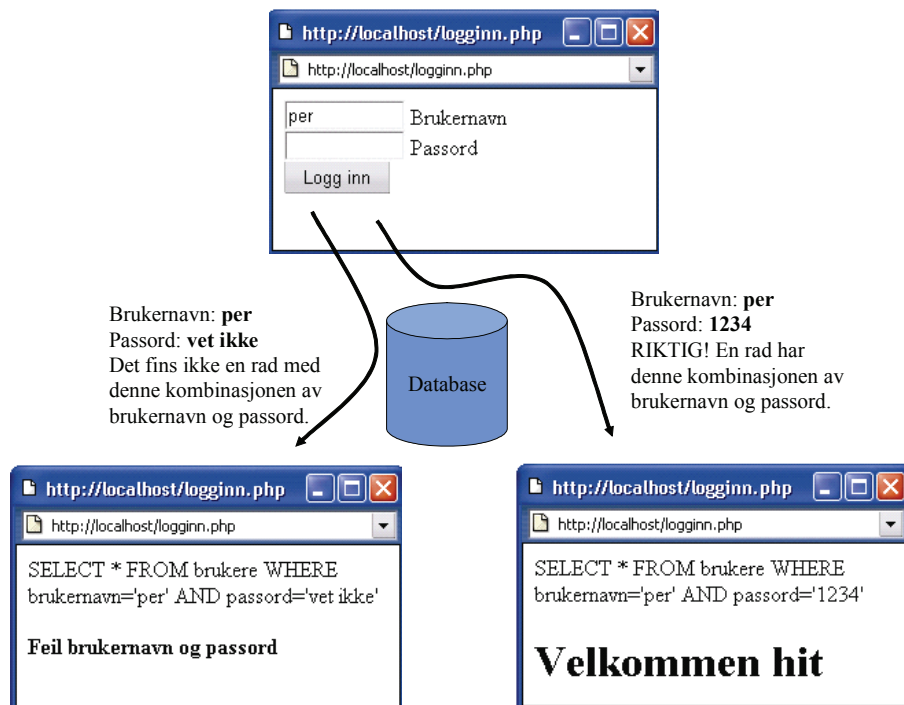
fall vises en feilmelding. Mange script er laget på denne måten, og logikken virker solid.

Kodesnutt 11.2 Skjema for innlogging og passordsjekk opp mot databasen

```
<?php
if ( !isset($_POST['knapp']) ) { //vis skjema
?>
  <form action='' method='post'>
  <input type='text' name='navn' size='12' /> Brukernavn<br />
  <input type='text' name='pass' size='12' /> Passord<br />
  <input type='submit' name='knapp' value='Logg inn' />
  </form>
<?php
} //if
else { //prosesser skjemaet
  $tilkobling = mysql_connect("localhost");
  mysql_select_db("prosjekt_database", $tilkobling);
  $sqlsetning = "SELECT * FROM brukere ";
  $sqlsetning .= "WHERE brukernavn='" . $_POST['navn'] . "' ";
  $sqlsetning .= "AND passord='" . $_POST['pass'] . "' ";
  echo $sqlsetning . "<p>";
  $resultat = mysql_query($sqlsetning, $tilkobling);
  $antall = mysql_num_rows($resultat);
  if ($antall >= 1) echo "<h1>Velkommen hit</h1>";
  else echo "<strong>Feil brukernavn og passord</strong>";
} //ferdig med prosessering
?>
```

Scriptet innebærer likevel store muligheter for en hacker. Hvorfor det, tror du? Innloggingsskjemaet er vist øverst i neste figur.

- Når Per skriver inn sitt brukernavn og feil passord, som vist til venstre, vil Per få beskjed om dette.
- Kun ved riktig kombinasjon av brukernavn og passord blir en rad funnet, med vellykket innlogging som resultat. Dette er vist til høyre.
- SQL-spørringen er skrevet ut for å poengtere at innskrevet brukernavn og passord er utgangspunkt for hvilken spørring som blir utført.



Figur 11.3 Spørringen er basert på innskrevet brukernavn og passord, og PHP-scriptet gir beskjed om feil eller riktig innlogging basert på resultatet.

Sannsynligheten er stor for at Ola Nordmann har et av disse brukernavnene: «ola», «olanor», «onordmann», «olan» og så videre. Tabellen `brukere` har både brukeren «ola» og brukeren «per». Hvor mange bedrifter har vel ikke en person som heter «Per» ansatt? Kombinert med vanlige navn og kontaktinformasjon oppgitt på hjemmesidene er det i de fleste systemer med brukerdatabaser lett å gjette seg til et eksisterende brukernavn.

Passordet er likevel ukjent, og det er dette sikkerheten i slike systemer belager seg på. Er det nok?

11.2.2 SQL-injection

Svaret ligger i luften, og er «nei». Det er ikke nok å basere seg på at passordet er ukjent. Det er nemlig ikke nødvendig for en hacker å vite passordet for å kunne logge seg inn i det systemet vi har gjennomgått på de foregående sidene. Forutsatt at en klarer å gjette seg til riktig brukernavn, kan en kreativ hacker med litt kjennskap til SQL enkelt formulere et brukernavn som vist i neste figur:

-----8<-----8<-----8<-----8<-----8<-----8<-----8<-----8<-----8<-----8<-----

Resten av kapittelet er kuttet bort, siden dette er et prøvekapittel. Det er 42 sider i alt i dette kapittelet i boka. De tema som blir tatt opp er: SQL-injection, privilegier i SQL, passord og sikkerhet, validering av inngangsdata, regulære uttrykk, adgangskontroll og autentisering, bruk av .htaccess, innstillinger og konfigurering av PHP, Apache og databasetjeneren, logging.