Sikker programmering i PHP

De fleste nystartede programmører tror at programmerings-sprog som PHP har indbygget sikkerhed. Dette er absolut ikke tilfældet. Der skal dog ikke meget til for at gøre din kode rimelig sikker...

Skrevet den **27. Feb 2009** af **flushout** I kategorien **Programmering / PHP** | ★★☆☆☆

Når man arbejder med web-programmering skal man være opmærksom på at fejl i en web-applikation kan skabe sikkerhedshuller i serveren.

I de senere år har der været en drastisk udvikling. For 5-10 år siden var 60-70% af sikkerhedshullerne på netværk/firewall siden og 30-40% var i web-applikationer. I dag er det omvendt ca. 65% af alle sikkerhedshuller ligger i web-applikationerne og omkring 35% i netværk/server.

Derfor er det vigtigt at kende nogle metoder til at sikre sine web-applikationer mod angreb. Jeg har herunder listet en række af de mest almindelige angreb og beskrevet hvordan man undgår dem.

CROSS SITE SCRIPTING (XSS):

XSS går kort sagt ud på at en hacker får mulighed for at køre scripts på din server. Det kan for eksempel være et script der sender brugerinformation på alle der besøger sitet videre til hackeren.

Al data der kommer fra querystrings, formularer, cookies osv. skal valideres. F.eks. skal et telefonnummer kun indeholde tal og der skal være 8 cifre.

Metoderne *htmlspecialchars()*, *htmlentities()* og *strip_tags()* skal benyttes på alle data som føres gennem systemet. Vær dog opmærksom på at htmlentities fjerner alle danske tegn (som f.eks. æ ø å) fra teksten.

Hvis man laver links eller benytter sig af querystrings bestående af dynamisk data skal *urlencode()* og *urldecode()* metoderne benyttes.

```
$tlf = htmlentities(urldecode($_GET['tlfnr']));

if (!eregi("^[0-9]{8}", $tlf)){
  echo = "Det indtastede tilfnummer: ".strip_tags($tlf)." er ikke gyldigt";
} else {
  echo "<a href=nextpage.php?tlf=".urlencode($tlf)."/>Gå til næste side</a>" ;
}
```

SQL INJECTION:

SQL injection opstår når en hacker får mulighed for at eksekvere kode på din database. Konsekvenserne ved et SQL injection angreb kan være lige fra at hele din database bliver downloadet til at hackeren får mulighed for at eksekvere filer på din webserver.

For at undgå SQL Injection skal *magic_quotes_gpc* være slået fra i din *php.ini* fil. Desuden bør alle DB

forespørgsler benytte *mysql_escape_string()* hvis der er dynamisk data indeholdt i forespørgslen. Desuden skal alt dynamisk indhold omsluttes af -> ´

```
$query= "SELECT * FROM users WHERE (tlfnummer='".mysql_escape_string($tlf)."')";
```

Man bør altid lave dobbelt tjek på om den nu også er den rigtige bruger der henter informationer. I eksemplet overfor vil man, som bruger kunne sende et telefonnummer afsted, som ikke er ens eget, og udfra det få alle informationer fra tabellen "users" om en anden bruger end en selv. Hvis man udvider det ovenstående kald med en "AND userID = $loggedInUser" vil man sikre sig mod det problem.

```
$query= "SELECT * FROM users WHERE (tlfnummer='".mysql_escape_string($tlf)."') AND (userID
='".mysql_escape_string($loggedInUser)."')";
```

COOKIE POISONING OG SESSION HIGHJACKING

Cookie poisoning og session highjacking går ud på at en hacker manipulerer eller stjæler dine cookies/sessions. Cookies og session id's benyttes ofte til at gemme på informationer om brugeren. Disse informationer kan give adgang til en hacker, hvis man ikke håndterer dem rigtigt.

Generelt set så gælder de samme regler som beskrevet under Cross Site Scripting, valider indholdet! Du bør dog kryptere indholdet af cookies og session id's hvis de indeholder informationer om den besøgende.

INITIALISER VARIABLE:

Det er fornuftigt at initialisere sine variable. Hvis *register_globals* ikke er slået fra i din *php.ini* fil initialiserer PHP automatisk variable. Hvis en given variabel ikke er blevet initialiseret før den bliver brugt kan der opstå situationer, hvor brugeren kan kaste en uønsket værdi på den variabel. F.eks. kan med en querystring kaste en værdi på en variabel som ikke bliver initialiseret. Forestil dig at $superuser variablen herunder ikke bliver initialiseret, så vil man i visse tilfælde kunne skaffe sig adgang blot ved at benytte en querystring: /filename.php?superuser=true

```
$superuser = false;
if( logged_in() ){
$superuser = true;
}
```

FILHÅNDTERING:

Include, require samt alle andre metoder til at åbne eller læse filer kan udnyttes hvis de åbnes vha. dynamisk data. I dette tilfælde bør man benytte sig af en kombination funktionerne *realpath()* og *basename()* for at sikre sig at det er den korrekte fil man er ved at åbne.

```
$filename = $_POST['filename'];
  $temp = basename(realpath($filename));
```

```
if ($filename != $temp){
   echo "BAD file!";
} else {
   echo "GOOD file!";
}
```

Desuden bør man ved fil-uploads benytte sig af funktionerne: *is_uploaded_file()* og *filesize()* for at sikre at filen der behandles rent faktisk er en uploaded fil og at filens størrelse er rigtig.

Det er en dårlig ide at oprette/uploade/kopiere en fil og først bagefter sætte adgangsrestriktioner på. Der kan opstå situationer, hvor en given bruger får adgang til filen i det øjeblik hvor den stadig ikke er skrivebeskyttet. Løsningen på dette er at benytte funktionen *umask()*, som sørger for at sætte adgangsrettighederne før filen oprettes/uploades/kopieres. Man skal dog være forsigtig med *umask()*, da man kan slette filer vha. denne funktion. Hvis man kalder *umask()* med dynamisk indhold er det derfor meget vigtigt at lave validering af input. Kig eventuelt på *chmod()* funktionen, den er i mange henseender et godt alternativ.

FORNUFTIGE LINKS:

Følgende sider indeholder supplerende oplysninger og dokumentation af nogle af de ting jeg ikke har gennemgået her.

- http://www.php.net/manual/en/[/b]
Online dokumentation for PHP - For at få yderligere information om alle de funktioner der er nævnt i denne artikel så søg efter funktionsnavnet på denne side.

- http://www.php.net/manual/en/ref.pcre.php[/b]
Documentation for regular expressions i PHP.

- http://phpsec.org/projects/guide/[/b]
Mere grundig gennemgang af sikkerhed i php.

- http://dk.php.net/manual/da/security.index.php[/b]
Mere grundig gennemgang af sikkerhed i php.

- http://www.webcafe.dk/artikler/apache/htaccesspassword/[/b]
En side med en grundlæggende beskrivelse af .htaccess og lidt om hvordan man krypterer.

- http://www.php.net/features.file-upload[/b]
God gennemgang af file-uploads, der er også eksempler på validering.

IØVRIGT:

Tak for kommentarerne! Jeg vil løbende forbedre denne artikel og lave nogle af de udvidelser og rettelser som i foreslår.

Jeg har valgt ikke at gennemgå de forskellige funktioner, de er beskrevet meget grundigt på php.net og andre sites og der er ingen grund til at jeg gentager noget der allerede er godt beskrevet.

Kommentar af philip d. 27. May 2005 | 1

Jeg er bestemt ikke enig i din påstand om at "htmlspecialchars(), htmlentities() og strip_tags()" skal køres på alt indhold. Det tjener i mine øjne ikke noget formål at fjerne HTML tags fra en string. Det giver ikke hackeren nogen form for kontrol. Dog skal de naturligvis fjernes hvis indholdet skal udskrives, men det er en anden sag.
Jeg syne sat addslashes() er et fint alternativ, samtidig med at det giver den fornødne sikkerhed, i hvert fald i forhold til SQL injections.
Artiklen er lidt overfladisk og beskriver ikke reelt hvad hackeren kan.

Kommentar af basementjack d. 10. May 2005 | 2

Virker grundigt for mig som nybegynder... Men mangler dog en grundigere forklaring af de mange forskellige funktioner der nævnes.

Kommentar af ducks d. 29. May 2005 | 3

Kommentar af mothail d. 11. May 2005 | 4

Lidt for overfladig...
Kommer med tips, men begrunder ikke vildt meget...

Kommentar af arkanoid d. 28. Jun 2005 | 5

Udmærket artikel, lidt kortfattet men gode tips til begynderne (hvis man tager sig tid til at læse dine links)

Kommentar af venchil d. 02. Jun 2005 | 6

Der mangler forklaring om, hvordan hackeren kan snige sig ind, hvad der gør ham i stand til det, og så synes jeg også der mangler en grundigere beskrivelse af, hvad de forskellige "funktioner" gør. Kan sagtens læse det i manualen men alligevel. Er begynder.

Kommentar af claus_joergensen d. 10. May 2005 | 7

Rimelig, dog mangler jeg en beskrivelse af regulær expressions (du bruger det i først eksempel, men forklare det ikke, preg_match() er også bedre og simplere), som alt for tit kan bruges til datavalidering. $_REQUEST bør man heller ikke bruge, brug $_GET eller $_POST alt efter hvad du ønsker. Hvis du bruger $_REQUEST kan jeg bruge en querystring istedet for en POST string, hvilket er usikkert.