Denne guide er oprindeligt
udgivet på Eksperten.dk

COMPUTERWORLD

Session-highjack...et halvt skridt mod en modgift

Denne artikel forsøger at beskrive hvad PHP session highjacking er og prøver at komme med et bud på hvordan man, i et html/php miljø via den alm. http protokol, delvis kan sikre sig mod dette.

Skrevet den **03. Feb 2009** af **johan.o** I kategorien **Programmering / PHP** | ★★☆☆☆

Hvad er en session

En PHP session er en mulighed for at overfører variabler fra et 'klient' request til et andet, uden at gøre indholdet af disse variabler synligt for omverdenen.

F.eks. besøger du én side hvor der defineres en variabel således :

```
<?php
$navn='Johan';
?>
```

Så er indholdet af $navn kun lig 'Johan' fra den tildeles værdien og indtil enden af scriptet er nået. Når du f.eks. klikker videre til den næste side er $navn 'tom' når scriptet startes, da serveren naturligvis ikke kan huske at $navn blev defineret ved et tidligere besøg - og sikke et held. Men til tider er det ønskværdigt at man kan 'knytte' indholdet af nogle variabler til en bestemt besøgende og så lade disse variabler følge den besøgende rundt på siderne.

Hvis du f.eks. ønsker at skrive en brugers navn øverst på alle de sider 'han' besøger på dit site. Jeg aner ikke hvorfor man skulle ønske dette, men blot for eksemplets skyld ;) I dette tilfælde ville det jo være rart at kunne 'overfører' en variabel med brugerens navn mellem alle de sider 'han' besøger, i stedet for at skulle spørge hver gang en side besøges.

Hertil kan sessions bruges.

Betragt dette lille stykke kode :

```
<?php
session_start();

$_SESSION[navn']="Johan";

echo "Navnet Johan er gemt i en session."; die;
?>
```

Det der sker i dette lille stykke kode er at der startes en session og session variablen 'navn' tildeles værdien 'Johan'. Men hvad indebærer det præcist ?

Den første gang du besøger en side hvor session_start() eksekveres, starter php en session og tildeler denne session et session_id. Forestil dig at serveren fremstiller en tom kasse hvor dette session_id skrives på. Når du så definerer session variablen 'navn' så gemmer serveren denne variabel i kassen med dit session_id på.

Derefter sendes der output fra serveren til klienten og 'skjult' i dette output sendes også dit session_id. Næste gang du så besøger en side på serveren sendes dit session_id tilbage til serveren og når linjen session_start() eksekveres så 'opdager' serveren at den har en kasse med dit session_id på og så indlæser den indholdet af den kasse, i vores tilfælde session variablen 'navn', igen.

Altså vil du efter ovenstående side er besøgt kunne besøge denne side og få flg. Output : "Dit navn er Johan.".

```
<?php
session_start();

echo "Dit navn er ".$_SESSION['navn']; die;
?>
```

Så i forståelsen af sessions er det værd at huske at alle session variabler ligger på serveren og det er kun session_id der 'flyver' frem og tilbage mellem server og klient.

Hvad er session highjacking ?

Det vil under normale omstændigheder kun være din browser og den server du er tilknyttet der kender dit session_id. Men da de informationer der flyder via http protokollen er i 'klar tekst' er det muligt for uvedkommende at opsnappe og læse dem. Jeg vil ikke begive mig ud i en forklaring af hvordan, primært fordi jeg ikke har undersøgt det nærmere men blot har konstateret at det er muligt. Så lad os forestille os at en 'bad guy' har aflyttet din trafik og har opfanget dit session_id, så vil det være en smal sag for ham at sende dette session_id til serveren som så vil finde kassen med dine variabler frem og stille dem til rådighed for ham.

Så er din session blevet highjacked. Du har ikke umiddelbart en chance for at se at denne 'bad guy' har adgang til dine variabler, så længe han ikke ændrer noget på hans vej.

Hvad kan jeg gøre

Lad mig først gøre det klart at der ikke findes en stensikker metode til at bevare absolut sikkerhed for at den bruger der sender et korrekt session_id til serveren, rent faktisk er den samme bruger som aktiverede dette session_id. Det skyldes, som tidligere nævnt, at i http protokollen er alt hvad der sendes mellem klient og server i 'klar tekst' og er derfor umiddelbart læsbart af enhver som har evnen til at sniffe på netværket. Det betyder at man, hvis man er lidt sortseer eller en fornuftig programmør, bør gå ud fra at alle 'transaktioner' mellem klient og server kan opfanges af en evt. 'lytter'.

I det følgende vil jeg forsøge at vise en fremgangs måde som i grove træk kun kan omgås hvis den uautoriserede 'lytter' har mulighed for, samtidig med sessionen highjack, at 'afbryde' den originale brugers adgang til det website der har givet ham det originale session_id.

Det følgende kræver en lidt bredere forståelse for bl.a. mySQL og PHP for at få det fulde udbytte og derudover er det værd at nævne, at der kan/bør tages yderligere skridt for at sikre koden end de viste, bl.a. bør mysql_real_escape_string() altid bruges i forbindelse med mySQL forespørgelser. Disse er udeladt

her for at bevarer en hvis grad af enkelthed i eksemplet.

Først sender vi, pr. request, følgende html-form til en bruger.

```
<form action="index.php" method="post">
<input type="text" name="user">
<input type="password" name="pwd">
<input type="submit" value="send">
</form>
```

Her skriver brugeren så sit brugernavn -> user og password -> pwd, som så sendes til serveren som POST-data.

På serveren, i index.php, bør der laves en validering af $_POST['user'] og $_POST['pwd'] f.eks. med preg_match men som tidligere nævnt vil jeg kun forholde mig til de elementer der kræves for at beskrive princippet.

De indtastede værdier sammenlignes med f.eks. en mySQL db.

```
<?php

session_start();

if(isset($_POST['user']) && isset($_POST['pwd']))
{
$q=mysql_query("SELECT * FROM tabel WHERE user='".$_POST[□user'].”' AND
pwd='".$_POST[□pwd'].”'");
if(mysql_num_rows($q)==1)
{
$_SESSION[□user']=$_POST[□user'];
echo "Du er nu logget ind."; die; } }

echo "Der kan være mange årsager, men du er ikke logget ind."; die;
?>
```

Således har vi nu, forhåbentligt, defineret $_SESSION['user'] variablen med et korrekt brugernavn og samtidig sendt et session_id() til den bruger som har afleveret det korrekte user/pwd.

Problemet er så at hvis der sidder en og 'lytter' med, kan vores session_id() opfanges og derved bruges til at 'snyde' sig til at serveren tror at 'lytteren' bør have adgang til $_SESSION['user'] variablen. Altså gøre det umuligt for serveren at skelne de to brugere fra hinanden.

Her kunne man så kaste sig over diverse header informationer, f.eks. user agent, i et forsøg på at sikre sig at de informationer som den originale bruger sendte til serveren i sin header er konstante, men i sidste ende kan 'lytteren' jo kopiere alle disse informationer.

I stedet forsøger vi at kontrollerer begivenhedernes gang ved at konstruerer en individuel 'kode' på serveren ved hvert besøg, som vi så sender tilbage til den bruger der har et valid session_id() og samtidigt gemmer vi denne kode i en tabel vi kalder online. Det er ofte et ønske at kunne vise hvor mange der er logget ind på diverse sites så hvis man i forvejen ønsker en sådan funktionalitet kan denne tabel med

fordel bruges, og hvis man som i dette eksempel bruger time() variablen behøver man heller ikke tilføje yderligere felter til sin tabel.

```
<?php

session_start();

if(isset($_POST['user']) && isset($_POST['pwd']))
{
$q=mysql_query("SELECT * FROM tabel WHERE user='".$_POST[□user'].
"' AND pwd='".$_POST[□pwd'].
"'");
if(mysql_num_rows($q)==1)
{
$_SESSION[□user']=$_POST[□user'];
$key=time();
setcookie("key", $key);
mysql_query("INSERT INTO online VALUES(□".$_POST[□user'].
"','".$key."')");
echo "Du er nu logget ind."; die; } }

echo "Der kan være mange årsager, men du er ikke logget ind."; die;
?>
```

Således tilføjes time() til vores online database og samtidigt sendes dette timestamp som en cookie til brugeren.

Derfor er det jo så også nødvendigt at checke, når brugeren sender sit næste request, at han er indehaver af den rigtigt cookie 'kode'.

Så følgende tilføjes til ovenstående.

```
if(isset($_SESSION[□user']) && isset($_COOKIE['key']))
{
$q=mysql_query("SELECT * FROM online WHERE user='".$_SESSION[□user'].
"' AND key='".$_COOKIE[□key'].
"'");
if(mysql_num_rows($q)==1)
{
$key=time();
mysql_query("UPDATE online WHERE user='".$_SESSION[□user'].
"' SET key='".$key."'");
setcookie("key", $key);
echo "Du er stadig logget ind."; die; } }
```

Således får vi nu sendt en ny 'kode' hver gang brugeren besøger siden. Hvis en anden har kopieret vores originale brugers session_id() og besøger vores side med dette session_id() vil han blive opfattet som den originale bruger men ligeså snart vores rigtige bruger besøger siden igen vil der være uoverensstemmelse mellem det der står i hans cookie og det der er i vores online tabel fordi den 'forkerte' bruger har aktiveret en udskiftning af denne kode. Derfor bør vi i dette tilfælde sørge for at session variablen og sessionen stoppes (Begge brugere logges af).

```
<?php
```

```
session_start();

if(isset($_SESSION[□user']) && isset($_COOKIE['key']))
{
$q=mysql_query("SELECT * FROM online WHERE user='".$_SESSION[□user']."' AND
key='".$_COOKIE[□key']."'");
if(mysql_num_rows($q)==1)
{
$key=time();
mysql_query("UPDATE online WHERE user='".$_SESSION[□user']."' SET key='".$key."'");
setcookie("key", $key);
echo "Du er stadig logget ind."; die;
}
else
{
mysql_query("DELETE FROM online WHERE user='".$_SESSION[□user']."'");
unset($_SESSION['user'];
session_destroy();
setcookie("key", "", time()-3600);
echo "En uoverensstemmelse har desværre gjort det nødvendigt at du logger på igen."; die; } }

if(isset($_POST['user']) && isset($_POST['pwd']))
{
$q=mysql_query("SELECT * FROM tabel WHERE user='".$_POST[□user']."' AND
pwd='".$_POST[□pwd']."'");
if(mysql_num_rows($q)==1)
{
$_SESSION[□user']=$_POST[□user'];
$key=time();
setcookie("key", $key);
mysql_query("INSERT INTO online VALUES(□".$_POST[□user']."','".$key."')");
echo "Du er nu logget ind."; die; } }

echo "Der kan være mange årsager, men du er ikke logget ind."; die;
?>
```

Det er naturligvis uheldigt at den 'forkerte' bruger rent faktisk har adgang til sitet så længe den 'rigtige' bruger IKKE bevæger sig rundt på sitet samtidigt og derfor er løsningen intet værd i det tilfælde hvor den 'forkerte' bruger har mulighed for at forhindre den 'rigtige' bruger i at besøge sitet efter at sessionen er highjacked.

Hvis man ønsker det kunne man også tilføje en mail-afsendelse til f.eks. en admin med besked om brugernavnet og tidspunktet for begivenheden.

Alle kode eksempler er primært vist for at hjælpe til forståelsen af princippet og bør ikke bruges i deres nuværende form, bl.a. på grund af andre sikkerhedshuller end de allerede nævnte og det skal da ikke afvises at der mangler et par , eller andre tegn hist og her :)

Håber artiklen har hjulpet nogen til lidt indblik i session begrebet og en af dets 'svagheder'.

Mvh. Johan

Kommentar af troxa d. 05. Feb 2006 | 1

Du har helt undgået at lave IP check mellem den rigtige bruger og serveren. Man kan selvfølgelig (hvis man er sej nok) lave sin ip om så det ser ud som om man kommer samme sted fra, men så er vi også ude i det ekstreme. Men hvis vi kender IP'en på den person der rigtigt er logget ind, så er det jo nemt at sige at alle andre ip'er end denne ikke er tilladt før der er logget ud (og så er den session død hvis man ellers kan kode ordentligt) :)

Kommentar af coder d. 08. Dec 2005 | 2

Det er sgu en god høker det der!
Og til jer der ønsker SSL - fuck om der er SSL på brugernavn og password... hvis det skal bruges skal der være SSL på under HELE forløbet.
Og Session-hijacking kan ikke kun ske med sniffing - husk det!!

Kommentar af cyberfinn d. 06. Dec 2005 | 3

En fin artikel. VIl dog mener, at man bør benytte en ssl-forbindelse, så snart man lave noget, hvor indholdet af en beskyttede siden, bør beskyttes ekstra godt

Kommentar af htx98i17 d. 09. Dec 2005 | 4

Kommentar af jakobdo d. 05. Feb 2006 | 5

Ok artikel, og gratis i dagens anledning.

Kommentar af jens_bach d. 06. Dec 2005 | 6

jeg er ennig i at man også skal bruge ssl
så snart der er et brugernavn/password så bør man bruge ssl

Kommentar af dustie d. 06. Dec 2005 | 7

Godt skrevet. Indrømmet, jeg har ikke rigtig læst den igennem, kun kigget lidt hist og her indtil videre, men det jeg har set ser rigtigt fint ud.

Kommentar af ekspertensbruger2004 d. 06. Dec 2005 | 8

Fin artikel og godt skrevet. Dog vil jeg mene, at du går fejl af din målgruppe. PHP-eksperter ved sandsynligvis allerede ovenstående og kan derfor ikke bruge den. Nybegynderne kan dog godt bruge det, men da du har udeladt vital kode bliver artiklen, som du selv siger, fyldt med sikkerhedsbrister. Disse vil nybegynderne ikke være i stand til at lukke uden yderligere hjælp.

Kommentar af simonhans73 d. 11. Feb 2006 | 9

hey go artikkel du har skrevet

Kommentar af there-is-only-xul d. 07. Dec 2005 | 10

jojo.. men denne artikel bygger på en bestemt måde at køre php på..
derudover vil man jo i et ordenligsystem ikke kunne bruge det til så meget, at hijacke en session ;) Men man kan jo evt. bruge RSA modeller hvis man ønsker det helt sikkert.. eller SSL som en del foreslår (men er i klar over hvad det koster ?).