

# Mengontrol Akses ke Data

**W**eb adalah sumber informasi global yang superluas. Setiap situs, termasuk situs Anda, merupakan bagian dari sumber data ini. Hanya saja, kadang-kadang kita tidak ingin semua orang dapat melihat data milik kita. Baik data yang sama, atau data lama. Di bab ini kita akan berbicara seputar caching, otentikasi, dan enkripsi. Kedua hal ini termasuk amat penting dipahami dalam operasi situs, dan keduanya membutuhkan pengetahuan mengenai protokol HTTP. Karena itu, kita juga akan membahas seluk-beluk protokol yang satu ini.

<b>Resep 4-1: Membuat Output Skrip Tidak Dapat di-Cache .....</b>	<b>48</b>
<b>Resep 4-2: Membuat Output Skrip Dapat di-Cache .....</b>	<b>50</b>
<b>Resep 4-3: Membuat Output Skrip Dapat di-Resume .....</b>	<b>53</b>
<b>Resep 4-4: Otentikasi dengan Sesi .....</b>	<b>57</b>
<b>Resep 4-5: Otentikasi dengan HTTP Basic Authentication .....</b>	<b>59</b>
<b>Resep 4-6: Otentikasi dengan Sesi dan HTTP .....</b>	<b>61</b>
<b>Resep 4-7: Otentikasi Sesi dengan “Remember Me” .....</b>	<b>62</b>

## Resep 4-1: Membuat Output Skrip Tidak Dapat di-Cache

Di antara skrip PHP di situs Anda dan pengunjung situs terdapat beberapa macam cache. Yang pertama adalah cache browser. Cache ini penting demi efisiensi koneksi pengunjung agar tidak perlu mengambil dokumen remote yang sama berulang kali dan menghabiskan bandwidth. Yang kedua adalah cache (satu atau lebih) proxy. Proxy adalah server yang berada sebagai penengah di antara user dan webserver. Biasanya ISP dan perusahaan memasang proxy. Cache pada proxy penting demi efisiensi perusahaan atau ISP agar beberapa pengunjung yang berbeda tapi mengunjungi halaman yang sama bisa berbagi dan tidak perlu masing-masing mengambil ulang. Yang ketiga adalah cache di mesin webserver, jika kebetulan webserver Anda menggunakan HTTP accelerator atau sistem cache ke file statik.

Ada beberapa alasan mengapa cache tidak diinginkan. Misalnya, dokumen yang dikembalikan bersifat pribadi atau rahasia, sehingga tidak baik tersimpan di disk proxy maupun browser. Bisa juga karena sedang ada penghitungan jumlah lewat counter dan banner, di mana setiap impresi atau halaman dikunjungi perlu menaikkan nilai counter. Atau, situs sering berubah, dan Anda tidak ingin orang meng-cache terlalu lama halaman-halaman basi.

Untuk memaksa agar skrip dipanggil dan dieksekusi ulang setiap kali ada kunjungan kembali ke URL skrip Anda, ada beberapa cara yang bisa dilakukan. Di bawah ini disebutkan cara-cara yang ada, disusun dari yang mulai paling “kurang ampuh” sampai yang “paling ampuh”. Perhatikan bahwa tidak selalu Anda harus menggunakan cara yang “paling ampuh”. Sebab, jika skrip Anda tidak dapat di-cache konsekuensinya adalah efisiensi menjadi rendah. Sehingga pemakai kesal karena terus-menerus harus menghubungi server, server menjadi berat karena dibanjiri request, dan bandwidth terboroskan. Gunakan cara seperlunya.

### Header Cache-Control: private

```
header("Cache-Control: private");
```

Dengan header ini, maka proxy akan mengerti bahwa dokumen yang Anda kembalikan hanya ditujukan bagi satu klien tertentu saja. Jadi jika user A mengambil `http://www.host.com/skrip.php` dan B juga mengambil `http://www.host.com/skrip.php`, proxy tidak akan meng-cache dokumen A bagi dokumen B, sehingga menjamin data tiap user tidak akan tertukar. Biasanya output skrip PHP memang tidak di-cache oleh proxy, karena tidak

mengandung header Last-Modified. Tapi seandainya ingin mengembalikan header tersebut, Anda bisa menggunakan header ini untuk memberitahu proxy. Jika menggunakan fasilitas sesi dari PHP, PHP akan otomatis mengirimkan header Cache-Control. Dengan header ini, dokumen tetap di-cache di browser.

### Header Cache-Control: no-cache dan Pragma: no-cache

```
header("Cache-Control: no-cache");
```

atau

```
header("Pragma: no-cache");
```

Kedua header ini sama artinya, namun ada sejumlah kecil proxy tua yang hanya mengenali Pragma. Agar lebih afdol, Anda bisa mengirimkan dua-duanya sekaligus. Header ini untuk melarang cache proxy meng-cache dokumen sama sekali. Dengan header ini, dokumen masih tetap bisa di-cache oleh browser.

### Header Expires dan Cache-Control: no-store, must-revalidate

```
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
```

```
header("Cache-Control: no-cache, no-store, must-revalidate");
```

Dengan mengirimkan header Expires bernilai tanggal di masa lalu, kita memberitahu proxy maupun browser bahwa dokumen yang Anda kembalikan telah kadaluarsa. Artinya, jangan simpan.

Dengan header ini, dokumen tidak disimpan di cache browser. Cocok jika halaman yang Anda kembalikan bersifat sensitif, misalnya untuk situs Internet Banking. Sebab jika dokumen di-cache browser, maka meskipun user telah logout, user lain pada komputer yang sama dapat menggali cache di browser dan memperoleh halaman tersebut.

Jika header Expires ini ditambah dengan header Cache-Control yang mengandung nilai no-store dan must-revalidate, maka selain tidak disimpan di cache disk browser, dokumen akan diminta ulang dari server, meskipun user menekan tombol Back browser. Artinya, pemakai lain maupun pemakai yang sama tidak bisa memanfaatkan tombol Back untuk mengintip isi dokumen. Penggunaan hanya jika perlu, karena bisa membuat kesal jika user harus sering melakukan Back.

Secara default, jika Anda menggunakan fasilitas sesi di PHP, header-header seperti inilah yang akan dikirim otomatis oleh PHP. Karena header-header ini merupakan antichache yang terampuh dan teraman.

## POST

Jika diakses melalui metode form POST, maka skrip Anda tidak akan di-cache oleh browser maupun proxy. Ini adalah salah satu perbedaan penting POST dan GET. Tentu saja, tidak semua skrip cocok diakses via POST. Skrip yang harus Anda sebutkan URL-nya dalam link <A HREF> misalnya, tidak bisa diklik menggunakan metode POST, kecuali menggunakan trik Javascript.

## Resep 4-2: Membuat Output Skrip Dapat di-Cache

Kebalikan dari resep sebelumnya, bagaimana jika Anda ingin agar output skrip dapat di-cache, baik oleh browser dan/atau oleh proxy? Ada beberapa alasan mengapa hal seperti ini perlu dilakukan. Pertama, jika ingin situs tampil lebih cepat. Jika situs terdiri dari banyak objek (gambar, javascript, dan sebagainya) maka semakin banyak objek yang ter-cache semakin sedikit sisa yang harus diambil dari network. Kedua, jika ingin mengirit bandwidth, baik bandwidth server maupun bandwidth user. Ketiga, jika ingin mengurangi beban server. Dengan semakin banyak orang yang mengambil halaman dari proxy, semakin banyak Anda dapat melayani pengunjung dengan spesifikasi mesin yang sama.

Di bawah ini beberapa cara melakukannya. Sebagian cara cocok pada situasi tertentu dan sebagian lagi tidak. Silakan dipilih atas pertimbangan sendiri.

### Menggunakan GET

Kalau ingin URL skrip Anda memungkinkan untuk di-cache, gunakan metode GET dan jangan POST. Tapi menggunakan GET saja tidak menjamin skrip Anda akan di-cache, bergantung pada faktor lain seperti setting browser dan proxy.

### Menggunakan PATH-INFO

Biasanya Anda menerima masukan parameter skrip dengan cara menyebutkannya di query string atau string setelah tanda tanya:

```
http://www.host.com/skrip.php?a=1&b=2
```

di mana PHP akan memberikan \$a bernilai 1 dan \$b bernilai 2 pada Anda (dan

juga \$\_REQUEST['a'] bernilai 1 serta \$\_REQUEST['b'] bernilai 2). Tapi kita bisa juga memanggil skrip dengan cara menyebutkan path info (string setelah tanda garis miring):

```
http://www.host.com/skrip.php/a=1;b=2
```

di mana \$a dan \$b kosong, tapi \$PATH\_INFO akan diset menjadi "/a=1;b=2" (tanpa kutip, diawali garis miring). Anda perlu memparse sendiri nilai a dan b.

Cara kedua—yakni menyertakan parameter di path info—kadang-kadang dipakai antara lain untuk memperlihatkan seolah-olah di server terdapat file statik (apalagi jika skrip.php diubah menjadi tanpa ekstensi skrip, sehingga seolah-olah tampak sebagai direktori). Cara ini juga biasanya dipakai untuk menghilangkan tanda tanya '?' agar tak perlu dipakai.

Tanda tanya biasanya pertanda skrip dinamik, sehingga banyak software cache proxy yang secara default tidak mau mengcachanya. Dengan menggunakan cara path info, skrip Anda tidak nampak sebagai skrip dinamik.

Dulu cara ini juga dipakai agar situs menjadi "search engine friendly" (ramah terhadap search engine), karena search engine "zaman dahulu" ogah mengikuti link yang mengandung tanda tanya. Tapi saat ini search engine seperti Google dan Altavista sudah mau mengikuti URL berquery string, jadi Anda tak perlu khawatir menggunakan cara query string jika alasannya hanya agar search engine friendly.

### Header Cache-Control: public

```
header("Cache-Control: public");
```

Header ini membantu memberitahu proxy bahwa dokumen dapat dicache. Header ini sifatnya opsional.

### Header Last-Modified

Apache biasanya memberikan header Last-Modified untuk file statik, yang diambil dari tanggal modifikasi file (mtime). Untuk skrip dinamik, defaultnya Apache dan PHP tidak mengirimkan header ini untuk Anda, karena memang tidak tahu hendak diambil dari mana (apakah dari field tanggal di database, dari file, dari sumber lain)? Jika memungkinkan/masuk akal, Anda bisa memberikan header ini untuk membantu cache proxy meng-cache dokumen Anda. Misalnya, jika membuat skrip *download wrapper* atau melewati download file ke skrip PHP dulu, Anda bisa melakukan seperti ini:

```
header("Last-Modified: " . gmdate("D, d M Y H:i:s",
filemtime($path)) . " GMT");
```

Perhatikan format tanggal yang diperlukan HTTP, yaitu dalam zona waktu GMT.

Jika sengaja ingin membuat skrip dapat di-cache, Anda dapat mengatur Last-Modified ke sebuah nilai di masa lalu:

```
header("Last-Modified: Mon, 26 Jul 1997 05:00:00 GMT");
```

### Header Expires

Header Expires juga bisa dipakai untuk memaksa dokumen di cache. Kebalikan dari di resep sebelumnya, untuk memaksakan cache kita mengatur nilai header ini menjadi tanggal di masa depan. Contohnya:

```
<?
// set dokumen kadaluarsa 30 menit dari sekarang
header("Last-Modified: Mon, 26 Jul 1997 05:00:00 GMT");
header("Expires: " . gmdate("D, d M Y H:i:s", time+30*60) . " GMT");
?>
```

Di resep di atas, kita juga menambahkan header Last-Modified agar lebih afdol.

Jika dokumen dikembalikan dengan menyertakan header Expires, maka cache proxy *tidak akan* mengambil ulang dokumen sampai tanggal kadaluarsa. Jadi meskipun user menekan *Refresh*, cache proxy akan tetap memberi hasil dari cache. Satu-satunya cara melewati cache ini adalah dengan mengubah string URL, misalnya: `http://www.host.com/skrip.php?a=1&b=2` menjadi `http://www.host.com/skrip.php?a=1&b=2&` atau `http://www.host.com/skrip.php?a=1&b=2&rand=23900242` atau `http://www.host.com/skrip.php?a=1&b=2&.92093`, dan seterusnya. Intinya, membuat string menjadi berbeda sehingga dokumen dianggap berbeda oleh cache tapi bagi skrip masukan parameter sama, sebab parameter ekstra yang ditambahkan tidak memiliki arti.

Mengapa jika kita berikan Expires bernilai tanggal di masa depan, cache jadi tidak mau melayani Refresh? Karena dengan memberi header Expires berarti kita memberi *jaminan* bahwa dokumen aman dipakai sampai tanggal kadaluarsa. Analogikan saat membeli susu atau daging kaleng, tanggal kadaluarsa berarti “aman dikonsumsi hingga tanggal ...”.

Kalau begitu apakah header ini berguna? Ya, header berguna jika Anda yakin dokumen memang tidak akan berubah dalam waktu tertentu. Contohnya adalah

ikon atau elemen layout yang bisa diberi Expires beberapa bulan ke depan. Ini bisa mengirit bandwidth cukup banyak.

Jika yang ingin di-cache adalah halaman dinamik, maka pastikan Anda memberikan nilai yang tidak terlalu jauh ke depan. Misalnya, halaman depan sebuah portal berita yang sibuk bisa diberi nilai Expires beberapa menit hingga setengah jam. Ini membantu mengurangi beban server, karena halaman depan amat banyak diakses dan di-refresh orang.

## Resep 4-3: Membuat Output Skrip Dapat di-Resume

Selain caching, resume adalah mekanisme untuk meningkatkan efisiensi. Karena koneksi Internet seringkali tidak stabil, maka demi efisiensi dalam mengambil resource yang besar. Seperti program berukuran puluhan MB atau bahkan CD image ratusan MB, HTTP mendukung resume. Artinya, jika sewaktu mendownload isi respon terputus di tengah-tengah, klien dapat meminta potongan berikutnya saja, tidak perlu menerima konten dari awal. Dengan resume bahkan sebuah file dapat diambil secara *swarming* atau potongan-potongan yang berbeda dari file yang sama diambil secara paralel untuk mempercepat transfer.

Mekanisme resume di HTTP dilakukan dengan header-header ini: Last-Modified, Range, Accept-Ranges, Content-Range. Berikut cara kerja resume. Pertama, klien melakukan request seperti biasa. Kedua, server memberi respon yang mengandung header respon Content-Length, Last-Modified, dan Accept-Ranges: bytes. Tanpa Content-Length, klien tidak bisa melakukan resume karena tidak tahu berapa panjang total respon. Tanpa Last-Modified pun, biasanya klien seperti GetRight dan Wget enggan melakukan resume karena tidak tahu apakah ketika direquest ulang konten telah berubah atau sama. Ketidakhadiran Content-Length dan Last-Modified merupakan salah satu ciri umum respon dinamik (yang umumnya tidak bisa diresume). Dan terakhir, Accept-Ranges dari server menyatakan bahwa server mendukung resume. Ketidadaan header ini membuat klien pun tidak akan “berani” melakukan resume.

```
GET /episodes/seinfeld_-_416_-_The_Outting.avi HTTP/1.1
Host: seinfeld.steven.localdomain

HTTP/1.1 200 OK
Content-Type: video/x-msvideo
```

```
Accept-Ranges: bytes
Content-Length: 54293657
Last-Modified: Thu, 16 Oct 2003 17:14:23 GMT
```

Respon dari Apache di atas mencirikan bahwa file bisa di-resume. Content-Length dan Last-Modified ada, Accept-Ranges juga ada. Maka seandainya setelah 100.000 byte selesai di-download koneksi putus, klien dapat mengirim request sebagai berikut:

```
GET /episodes/seinfeld_-_416_-_The_Outting.avi HTTP/1.1
Host: seinfeld.steven.localdomain
Range: bytes=100000-

HTTP/1.1 206 Partial Content
Content-Type: video/x-msvideo
Content-Length: 54193657
Accept-Ranges: bytes
Content-Range: bytes 100000-54293656/54293657
Last-Modified: Thu, 16 Oct 2003 17:14:23 GMT
```

Perhatikan respon dari Apache bukan berstatus 200 melainkan 206. Karena sama-sama kepala 2xx, maka request juga berhasil, tapi 206 menandakan respon hanya sebagian saja yang dikembalikan, disesuaikan dengan permintaan.

Perhatikan juga bahwa klien meminta dari byte ke-100.000 karena byte 0-99.999 telah diambil. Perhitungan offset dimulai dari 0. Di sisi respon, Apache memberitahu bahwa panjang konten (Content-Length) kini tinggal 54.193.657 (54.293.657 dikurangi 100.000) dan rentang yang dikembalikan (Content-Range) adalah dari byte 100.000 hingga offset terakhir.

Dengan demikian, seandainya putus kembali di bagian lain, klien bisa meminta range yang sesuai dan Apache akan mengembalikan potongan yang belum ter-download. Bahkan, program download seperti Flashget melakukan swarming, yaitu dengan mengirim beberapa request berbeda range sekaligus. Mis: file berukuran 10.000 diambil dalam 5 bagian secara paralel dengan masing-masing header request Range sbb: 0-1999, 2000-3999, 4000-5999, 6000-7999, 8000-9999. Nanti Flashget yang akan menyusun kembali potongan tiap respon menjadi satu file utuh. Berbekal pengetahuan ini, kita bisa membuat skrip yang mendukung resume. Ini berguna misalnya saat kita membuat skrip *wrapper* untuk download—user ingin

mendownload file, tapi kita paksa melewati skrip PHP kita dulu agar bisa kita cek cookie/user/passwordnya, atau dicatat hit dan statistiknya, atau diberlakukan pembatasan bandwidth, dan sebagainya. Kalau file yang ingin di-download besar, tentu membantu jika skrip PHP tersebut mendukung resume. Berikut ini contoh kode untuk melakukannya:

```
1|<?
2|
3|//
4|// download-resume.php
5|//
6|
7|$download_dir = "/home/steven/files";
8|
9|# saring karakter berbahaya, terutama '/' atau '\'
10|$file = preg_replace("/[^A-Za-z0-9_.-]+/", "", $PATH_INFO);
11|
12|$path = "$download_dir/$file";
13|
14|if (!$file || !file_exists($path)) {
15|    header("HTTP/1.1 404 Not Found");
16|    echo "<h1>404 Tidak Ditemukan!</h1>";
17|    exit;
18|}
19|
20|$size = filesize($path);
21|$lastmod = "Last-Modified: ". gmdate("D, d M Y H:i:s",
22|                                     filetype($path)) . "
23|    GMT";
24|
25|if (!$fp = fopen($path, "r")) {
26|    header("HTTP/1.1 500 Internal Server Error");
27|    echo "<h1>505 Tidak Bisa Didownload!</h1>";
28|    exit;
29|}
30|$start = 0; $end = $size-1;
31|
32|if (isset($_ENV['HTTP_RANGE']) &&
33|    preg_match('/^bytes=(\d+)-(\d*)/', $_ENV['HTTP_RANGE'],
34|               $m)) {
```

```

34| $start = $m[1]; if ($m[2]) $end = $m[2];
35| $x=1;
36|}
|
38|if ($start > $end || $start >= $size) {
39| header("HTTP/1.1 416 Requested Range Not Satisfiable");
40| header("Content-Length: 0");
41| header($lastmod);
42| exit;
43|}
|
45|if ($start==0 && $end==$size) {
46| header("HTTP/1.1 200 OK");
47|} else {
48| header("HTTP/1.1 206 Partial Content");
49| header("Content-Range: bytes $start-$end/$size");
50|}
|
52|$left = $end-$start+1;
|
54|header("Content-Type: application/octet-stream");
55|header($lastmod);
56|header("Accept-Ranges: bytes");
57|header("Content-Length: $left");
|
59|fseek($fp, $start);
60|while ($left > 0) {
61| $bytes = $left > 8192 ? 8192 : $left;
62| echo fread($fp, $bytes);
63| $left -= $bytes;
64|}
|
66|fclose($fp);
|
68|?>

```

Pada skrip di atas, mula-mula diambil nama file yang disebutkan via `PATH_INFO` yang dipilih agar nanti bisa menyebutkan URL download seperti `http://steven.localdomain/download.php/SOMEFILE-1.2.BIN`. Bukan `http://steven.localdomain/download.php?file=SOMEFILE-1.2.BIN`. Prefiks “/” dan karakter berbahaya lain disaring dengan regex (baris 10) sehingga `$file` kini berisi

"SOMEFILE-1.2.BIN".

Lalu keberadaan file diperiksa (baris 14-18). Jika file tidak ada, kita mengembalikan status 404. Lalu di baris 24-28 juga perlu dicek apakah file benar-benar bisa dibuka dengan `fopen()`. Jika tidak bisa, kembalikan status 500 (kesalahan pada server). Mungkin setting permission di server tidak benar sehingga file gagal dibuka.

Kemudian di baris 32-36 kita memeriksa header request Range (yang disediakan oleh PHP di variabel `$_ENV['HTTP_RANGE']`). Jika ternyata klien meminta range tertentu, kita sesuaikan `$start` dan `$end` (defaultnya `$start` adalah 0 dan `$end` adalah offset terakhir file).

Lalu baris 38-43 kita mengecek nilai `$start` dan `$end`. Jika di luar rentang file, maka kembalikan status 416. Status ini berarti rentang yang diminta tidak dapat dipenuhi. Kepala 4xx artinya kesalahan pada pihak klien.

Setelah semua oke, kita bisa mulai mengembalikan respon. Pertama-tama cetak baris status yang sesuai, 200 atau 206 (baris 45-50). Lalu cetak header-header yang diperlukan—yang mengindikasikan kita memang mendukung resume dan mengembalikan hanya sebagian respon—di baris 54-57.

Anda bisa memodifikasi skrip ini untuk menambahkan pengecekan cookie misalnya, atau header Referer, atau menambahkan `sleep()` di dalam loop `while` untuk membatasi bandwidth.

## Resep 4-4: Otentikasi dengan Sesi

```

1|<?
|
3|//
4|// otentikasi-sesi-basic.php
5|//
|
7|session_register('auth_user');
|
9|if (isset($logout)) {
10| session_destroy();
11| echo "

```

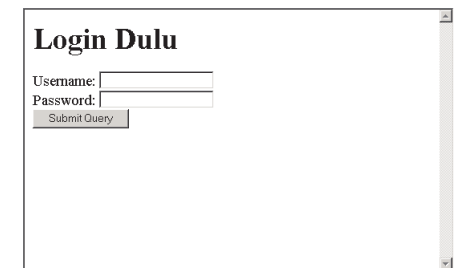
```

12| <p>Anda telah logout.
13| <p><a href=$PHP_SELF?".rand().">Klik di sini untuk login
    lagi</p>
14| ";
15| exit;
16|}
17|
18|if (!isset($auth_user)) {
19|    if (isset($user) && isset($pass)) {
20|        // cek apakah user dan password betul
21|        if ($user == "jerry" && $pass == "comedian") {
22|            // betul, set $auth_user
23|            $auth_user = "jerry";
24|        } else {
25|            // salah, tampilkan pesan salah
26|            echo "User/password salah, tekan Back untuk
    mengulangi!";
27|        }
28|    } else {
29|        // tidak ada user/pass yang disubmit dari form, tampilkan
    form
30|        echo
31|        "<h1>Login Dulu</h1>
32|        <form method=POST>
33|            Username: <input name=user><br>
34|            Password: <input name=pass type=password><br>
35|            <input type=submit><br>
36|        </form>
37|        ";
38|    }
39|}
40|if (!isset($auth_user)) exit;
41|
42|// tampilkan dokumen seperti biasa
43|echo "<p>Halo, $auth_user!</p>";
44|echo "<p><a href=$PHP_SELF?".rand().">Halaman selanjutnya</
    a></p>";
45|echo "<p><a href=$PHP_SELF?logout=1>Logout</a></p>";
46|
47|?>

```

Resep ini adalah pola dasar otentikasi menggunakan sesi. Pertama kali seorang user mengunjungi situs, nilai variabel sesi `$auth_user` belum diset sehingga user akan masuk ke baris 19. Skrip lalu mengecek apakah variabel form `$user` dan `$pass` diset. Jika belum ada, karena memang belum ada form HTML yang dicetak, maka cetak dulu di baris 30-37. Lihat Gambar 4-1. Jika form telah di-submit, maka cek dengan nilai yang benar. Baris 21 dapat diganti pengecekan ke database MySQL, file teks, atau database lainnya. Jika user dan password benar, set nilai variabel sesi `$auth_user`. Nilai ini akan terus diset di halaman berikutnya sampai browser ditutup, sehingga skrip tidak perlu mengecek user atau password ke database berulang-ulang. Baris 40 memastikan bahwa jika `$auth_user` tetap belum ada, user tidak diijinkan melihat dokumen yang diinginkan.

Potongan kode untuk pengecekan seperti yang ada di baris 18-40 perlu ada di setiap halaman yang ingin diproteksi. Anda dapat menaruhnya di sebuah file tersendiri yang nanti di-include() dari setiap file lain. Untuk melakukan logout (baris 9-16) kita bisa memanggil `session_destroy()` untuk melupakan sesi user, sehingga semua variabel sesi menjadi terreset kembali.



Gambar 4-1. Halaman Login.

## Resep 4-5: Otentikasi dengan HTTP Basic Authentication

Otentikasi ini kadang-kadang dijuluki orang “otentikasi pop up” atau “otentikasi dialog box” karena skrip kita tidak meminta user dan password lewat form HTML, melainkan melalui kotak dialog atau window pop up. Lihat Gambar 4-2.

```

1|<?
2|
3|//
4|// otentikasi-http.php
5|//
6|
7|$authenticated = 0;
8|
9|if (isset($_SERVER['PHP_AUTH_USER']) &&

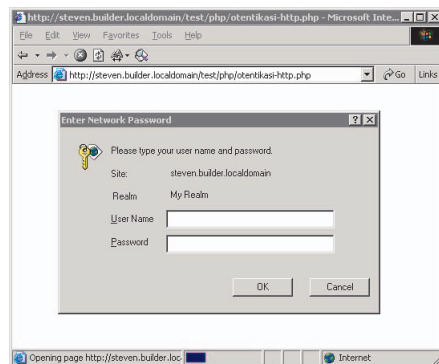
```



```

10| isset($_SERVER['PHP_AUTH_PW'])) {
11|
12| // cek user dan password
13| if ($_SERVER['PHP_AUTH_USER'] == "jerry" &&
14|     $_SERVER['PHP_AUTH_PW'] == "comedian") {
15|     // benar
16|     $authenticated = 1;
17| }
18|}
19|
20| if (!$authenticated) {
21|     header('WWW-Authenticate: Basic realm="My Realm"');
22|     header('HTTP/1.0 401 Unauthorized');
23|     echo
24|         "<h1>Forbidden</h1>
25|         Anda dilarang masuk!
26| ";
27|     exit;
28|}
29|
30| echo "
31| <p>Hallo {$_SERVER['PHP_AUTH_USER']}.</p>
32| <p>Senang melihat Anda lagi.</p>
33| <a href=$PHP_SELF?".rand().">Halaman selanjutnya</a>
34| ";
35|
36| ?>

```



Gambar 4-2. Pop up Otentifikasi.

Resep di atas adalah kerangka untuk melakukan otentikasi HTTP. Baris 13-14 dapat Anda ganti dengan pengecekan user dan password yang sebenarnya. Baris 30 dan seterusnya dengan isi dokumen yang ingin diproteksi.

Sayangnya, dengan otentikasi dengan HTTP seperti ini kita sulit melakukan logout. Satu-satunya cara yang pasti untuk logout adalah dengan menutup browser. Namun keunggulan otentikasi dengan cara ini adalah dapat bekerja dengan klien-klien nonbrowser seperti Wget.

## Resep 4-6: Otentikasi dengan Sesi dan HTTP

Resep ini adalah gabungan 2 resep sebelumnya. Dengan kode berikut, user dan password diminta secara HTTP ("pop up") tapi setelah itu kita mengeset variabel sesi `$auth_user` agar tidak perlu berulang mengecek user beserta password tiap request.

```

1| <?
2|
3| //
4| // otentikasi-sesi-http.php
5| //
6|
7| session_register('auth_user');
8|
9| if (!isset($auth_user)) {
10|     if (isset($_SERVER['PHP_AUTH_USER']) &&
11|         isset($_SERVER['PHP_AUTH_PW'])) {
12|
13|         // cek user dan password
14|         if ($_SERVER['PHP_AUTH_USER'] == "jerry" &&
15|             $_SERVER['PHP_AUTH_PW'] == "comedian") {
16|             // benar
17|             $auth_user = $_SERVER['PHP_AUTH_USER'];
18|         }
19|     }
20| }
21|
22| if (!isset($auth_user)) {
23|     header("WWW-Authenticate: Basic realm=\"My Realm\"");

```



```

24| header("HTTP/1.0 401 Unauthorized");
25| echo
26|     "<h1>Forbidden</h1>"
27|     "Anda dilarang masuk!"
28|     ";
29| exit;
30|}
31|
32|echo "
33| <p>Hallo {$SERVER['PHP_AUTH_USER']}</p>
34| <p>Senang melihat Anda lagi.</p>
35| <p><a href=$PHP_SELF?.rand().>Halaman selanjutnya</a></p>
36|";
37|
38|?>

```

## Resep 4-7: Otentikasi Sesi dengan “Remember Me”

Resep di bawah adalah modifikasi resep 4-3 dengan tambahan fitur yang sering dimiliki oleh banyak situs forum atau portal yaitu “Remember Me” (atau “Log Me In Automatically”). Cara melakukan hal ini yaitu dengan menyimpan identitas \$auth\_user di cookie persisten. Pada resep, cookie yang dipakai bernama auth\_user dan diset kadaluarsa 1 tahun (baris 10). Cookie bernama auth\_check juga dikirim sebagai nilai rahasia yang unik untuk setiap nilai \$auth\_user; cookie ini dipakai sebagai pengaman untuk mencegah user yang nakal memasukkan nilai cookie sembarang auth\_user (misalnya: admin atau root atau user lain). Tanpa mengetahui nilai auth\_check, kita tidak akan menerima nilai auth\_user yang diberikan oleh user lewat cookie. Saat logout, selain menghapus sesi kita juga perlu menghapus cookie persisten auth\_user (baris 16).

```

1|<?
2|
3|//
4|// otentikasi-sesi-remember.php
5|//
6|
7|$PASSWORDS["jerry"] = "comedian";
8|$PASSWORDS["kramer"] = "giddy up!";

```

```

9|$SECRET = "19023cz9x80z98xc9w";      # ubah sebelum pakai!
10|$MEMORY = 365*24*3600;                # 1 tahun
11|
12|session_register('auth_user');
13|
14|if (isset($logout)) {
15|    session_destroy();
16|    setcookie("auth_user", "");
17|    echo "<p>Anda telah logout.</p>";
18|    echo "<p><a href=$PHP_SELF?.rand().>Klik untuk login
19|        lagi</p>";
20|    exit;
21|}
22|
23|if (!isset($auth_user)) {
24|    if (isset($user) && isset($pass)) {
25|        // cek apakah user dan password betul
26|        if (isset($PASSWORDS[$user]) && $PASSWORDS[$user] ==
27|            $pass) {
28|            $auth_user = $user;
29|            $now = time();
30|            if (isset($remember)) {
31|                setcookie("auth_user", $user, $now+$MEMORY);
32|                setcookie("auth_check", md5($user . $SECRET),
33|                    $now+$MEMORY);
34|            }
35|        } else {
36|            echo "User/password salah!";
37|        }
38|    } elseif (isset($_COOKIE['auth_user']) &&
39|        isset($_COOKIE['auth_check'])) {
40|        // cek cookie
41|        if ($_COOKIE['auth_check'] ==
42|            md5($_COOKIE['auth_user'].$SECRET)) {
43|            // benar
44|            $auth_user = $_COOKIE['auth_user'];
45|            $now = time();
46|            // perpanjang masa expire cookie
47|            setcookie("auth_user", $_COOKIE['auth_user'],
48|                $now+$MEMORY);
49|            setcookie("auth_check", $_COOKIE['auth_check'],

```

#### Resep 4-7: Otentikasi Sesi dengan "Remember Me"

```
$now+$MEMORY);  
45|     } else {  
46|         echo "Cookie tidak valid!";  
47|     }  
48| }  
49|}  
  
51| if (!isset($auth_user)) {  
52|     echo "  
53|         <h1>Login Dulu</h1>  
54|         <form method=POST>  
55|         Username: <input name=user><br>  
56|         Password: <input name=pass type=password><br>  
57|         <input type=checkbox name=remember>Remember me<br>  
58|         <input type=submit><br>  
59|         </form>";  
60|     exit;  
61| }  
  
63| // tampilkan dokumen seperti biasa  
64| echo "<p>Halo, $auth_user!</p>";  
65| echo "<p><a href=$PHP_SELF?logout=1>Logout</a></p>";  
  
67| ?>
```