PHPのセッションに関するよしなしごと

１．セッションとは？
　PHPのセッションとは、いわゆる「HTTPセッション」のことであり、その実体はクッキーの一種で、目的はブラウザと鯖の間である一定期間、値を連携させるための仕掛けです。
　全然分からないと思いますので、んでは実体であるところの「クッキー」から見ていきます。

２．クッキーとは？
　ブラウザと鯖の間で値をやりとりするには、以前は「GET」と「POST」という二種類しかありませんでした。どういうやり方かというと、ブラウザから一方的にGETあるいはPOSTというHTTPで値を鯖に知らせるやり方です。んで、鯖はその値をパラメータとしてCGIを動かして、処理するわけです。
　ところが、このやりかたの場合ブラウザ→鯖の一方通行だったわけです。
　どーにかして、鯖→ブラウザ方向も使えるようにしたかったわけです。(HIDDENフォームを使ってCGI側で、ブラウザ側から送信されたGET/POST変数を埋め込む方法もある。けど、めんどい。)
　そこで、HTTPのヘッダー部分を拡張し、「Set-Cookie」と「Cookie」という二つのフィールドを追加しました。
「Set-Cookie」：鯖→ブラウザ
「Cookie」：ブラウザ→鯖
　これで、結構楽に鯖とブラウザの間で値を一定期間保持できるようになったわけです。

３．実際にクッキーのやりとりを見てみませう。

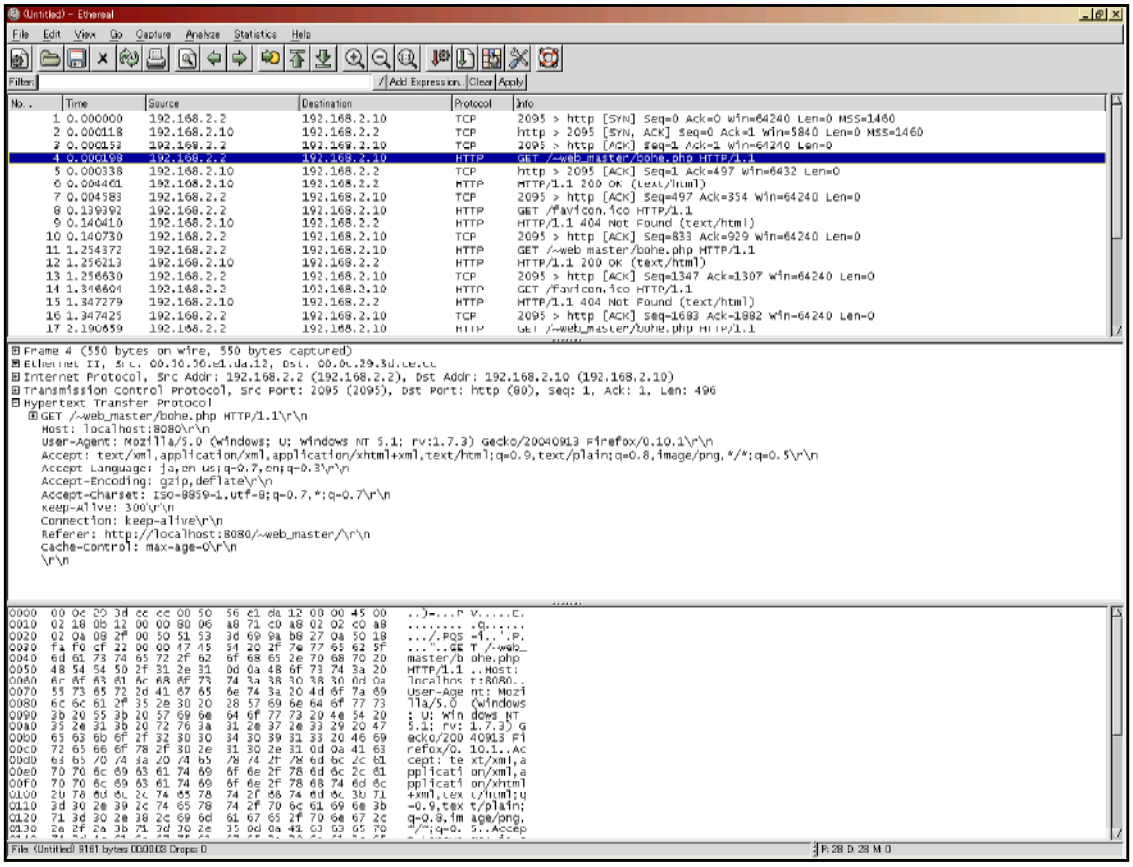
ＰＨＰソースコード：bohe.php
```
<?php
ob_start();
if(!isset($HTTP_COOKIE_VARS["a"])) setcookie("a", 1);
else setcookie("a", ++$HTTP_COOKIE_VARS["a"]);
if(!isset($HTTP_COOKIE_VARS["b"])) setcookie("b", 1);
else setcookie("b", ++$HTTP_COOKIE_VARS["b"]);

print_r($HTTP_COOKIE_VARS);
?>
```

中身はなんてーことはありません。クッキーにセットされてる値を1で初期化して、以降、+1してひょうじするだけです。

HTTPのやりとりにはEthereal をつかってパケットをキャプチャしてみました。フィルタリングは
tcp port 80
で結構です。

実況中継します。

①ブラウザ側
最初のリクエストです。

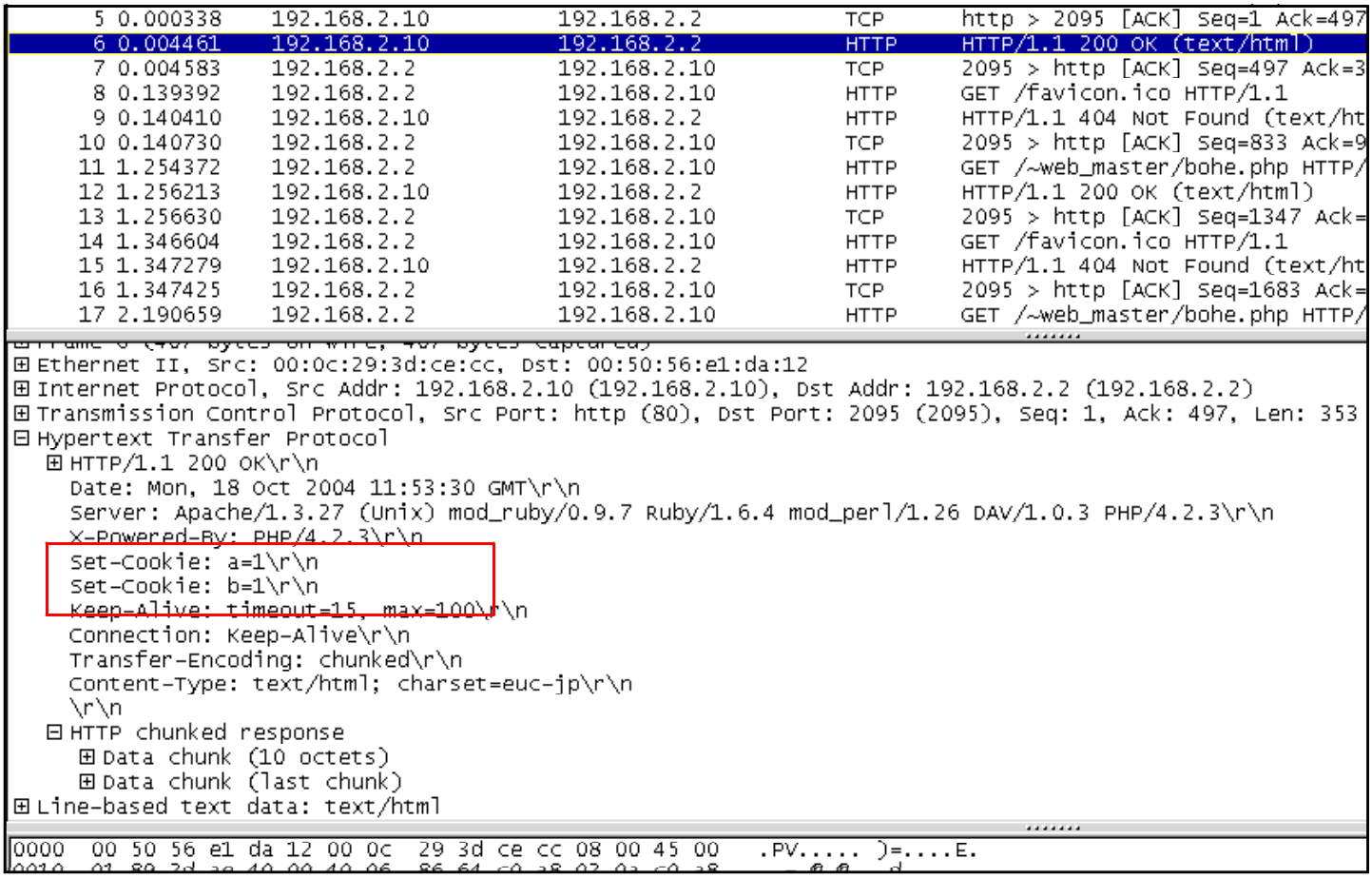


もうちょっと主要部分を拡大してみます。

```
     4 0.000198    192.168.2.2     192.168.2.10    HTTP    GET /~web_master/bohe.php HTTP/1.1
     5 0.000338    192.168.2.10    192.168.2.2     TCP     http > 2095 [ACK] Seq=1 Ack=497 Wir
     6 0.004461    192.168.2.10    192.168.2.2     HTTP    HTTP/1.1 200 OK (text/html)
     7 0.004583    192.168.2.2     192.168.2.10    TCP     2095 > http [ACK] Seq=497 Ack=354 W
     8 0.139392    192.168.2.2     192.168.2.10    HTTP    GET /favicon.ico HTTP/1.1
     9 0.140410    192.168.2.10    192.168.2.2     HTTP    HTTP/1.1 404 Not Found (text/html)
    10 0.140730    192.168.2.2     192.168.2.10    TCP     2095 > http [ACK] Seq=833 Ack=929 W
    11 1.254372    192.168.2.2     192.168.2.10    HTTP    GET /~web_master/bohe.php HTTP/1.1
    12 1.256213    192.168.2.10    192.168.2.2     HTTP    HTTP/1.1 200 OK (text/html)
    13 1.256630    192.168.2.2     192.168.2.10    TCP     2095 > http [ACK] Seq=1347 Ack=1307
    14 1.346604    192.168.2.2     192.168.2.10    HTTP    GET /favicon.ico HTTP/1.1
    15 1.347279    192.168.2.10    192.168.2.2     HTTP    HTTP/1.1 404 Not Found (text/html)
    16 1.347425    192.168.2.2     192.168.2.10    TCP     2095 > http [ACK] Seq=1683 Ack=1882
    17 2.190659    192.168.2.2     192.168.2.10    HTTP    GET /~web_master/bohe.php HTTP/1.1
⊞ Frame 4 (550 bytes on wire, 550 bytes captured)
⊞ Ethernet II, Src: 00:50:56:e1:da:12, Dst: 00:0c:29:3d:ce:cc
⊞ Internet Protocol, Src Addr: 192.168.2.2 (192.168.2.2), Dst Addr: 192.168.2.10 (192.168.2.10)
⊞ Transmission Control Protocol, Src Port: 2095 (2095), Dst Port: http (80), Seq: 1, Ack: 1, Len: 496
⊟ Hypertext Transfer Protocol
  ⊞ GET /~web_master/bohe.php HTTP/1.1\r\n
    Host: localhost:8080\r\n
    User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; rv:1.7.3) Gecko/20040913 Firefox/0.10.1\r\n
    Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;c
    Accept-Language: ja,en-us;q=0.7,en;q=0.3\r\n
    Accept-Encoding: gzip,deflate\r\n
    Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
    Keep-Alive: 300\r\n
    Connection: keep-alive\r\n
    Referer: http://localhost:8080/~web_master/\r\n
    Cache-Control: max-age=0\r\n
    \r\n
```

普通のGETメソッドによるHTTPヘッダーです。
では、それに対する前掲PHPの応答です。

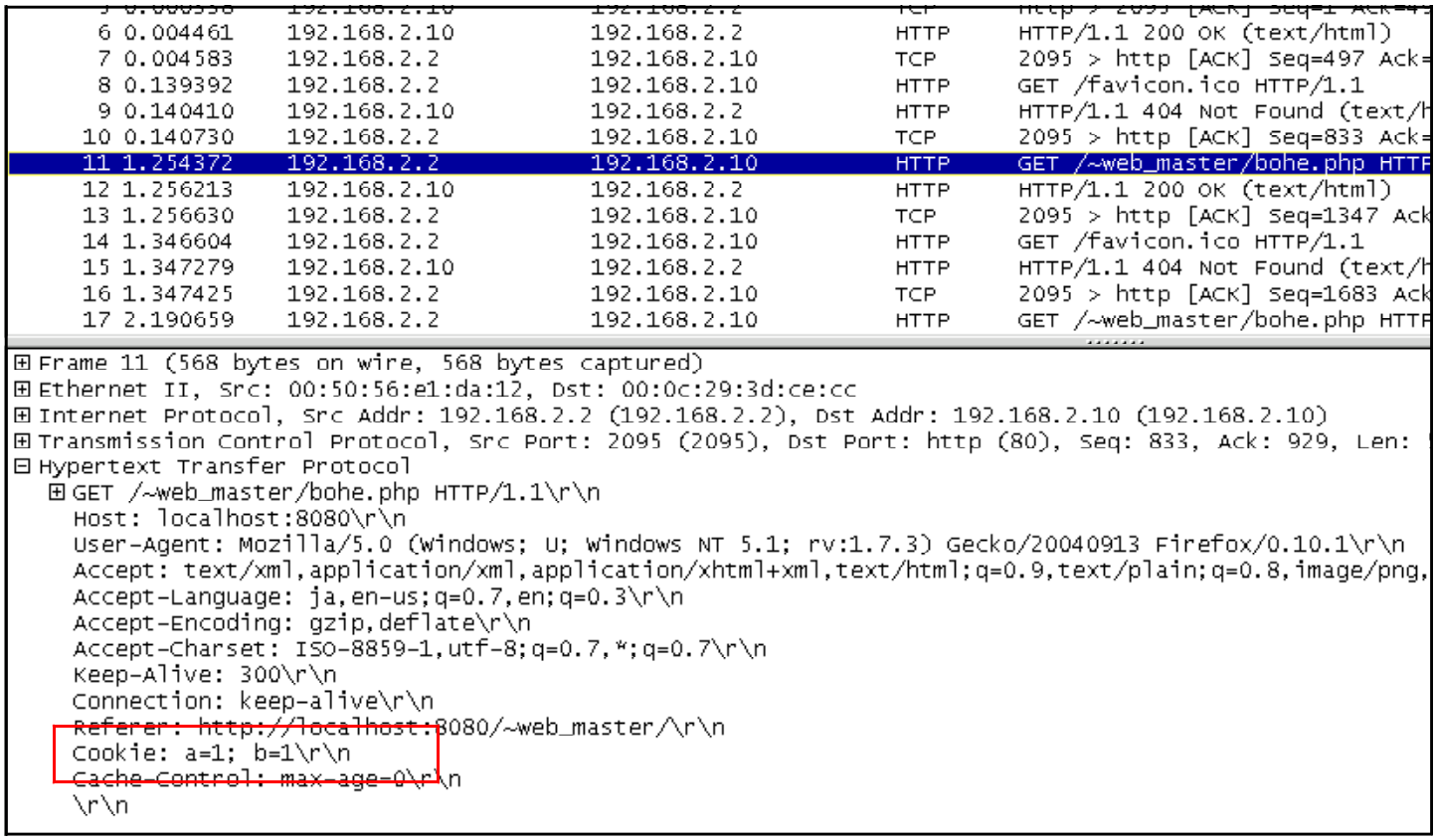
②鯖応答



はい。「Set-Cookie」ヘッダーがしっかりと出現しています。対応するPHPコードが
if(!isset($HTTP_COOKIE_VARS["a"])) setcookie("a", 1);
と
if(!isset($HTTP_COOKIE_VARS["b"])) setcookie("b", 1);
です・・・たぶん。いえ、ちょっとこのスクリプト、実際のとは違うんです。このキャプチャで表示されてるスクリプト、消しちゃって・・・(汗)。んで、確かこんな感じだったなーみたいに作ったのが載せてるのです。
でもでも、これ、1で初期化せずにいきなり飛んで2から始まったりするんです。・・・ま、まあ「理想的な」スクリプトだったら、このキャプチャ画面の通りになるということで。

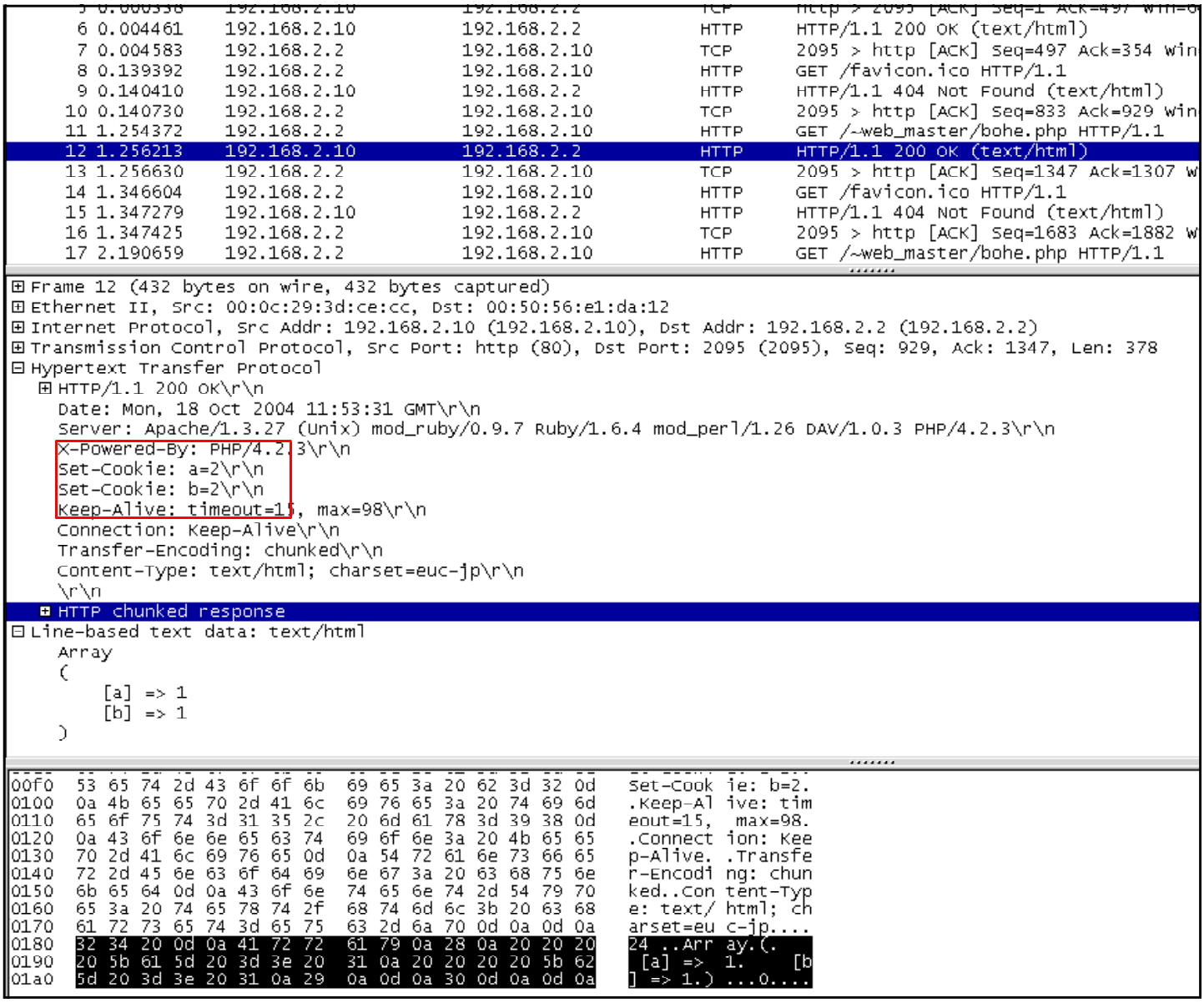
ともかく、これでブラウザ側に「クッキー」がセットされたわけです。では、リロードさせてみます。
リロードするとき、ブラウザはどんなHTTPヘッダーを鯖に送信するのか？

③も一度ブラウザから・・・の送信。



「Cookie」ヘッダーがちゃんとついてますね。
しかも、セミコロンで区切って「変数名=値」形式で送信されてます。
んではんでは、これに対する応答はどうなるのか？

④鯖の応答二度目。

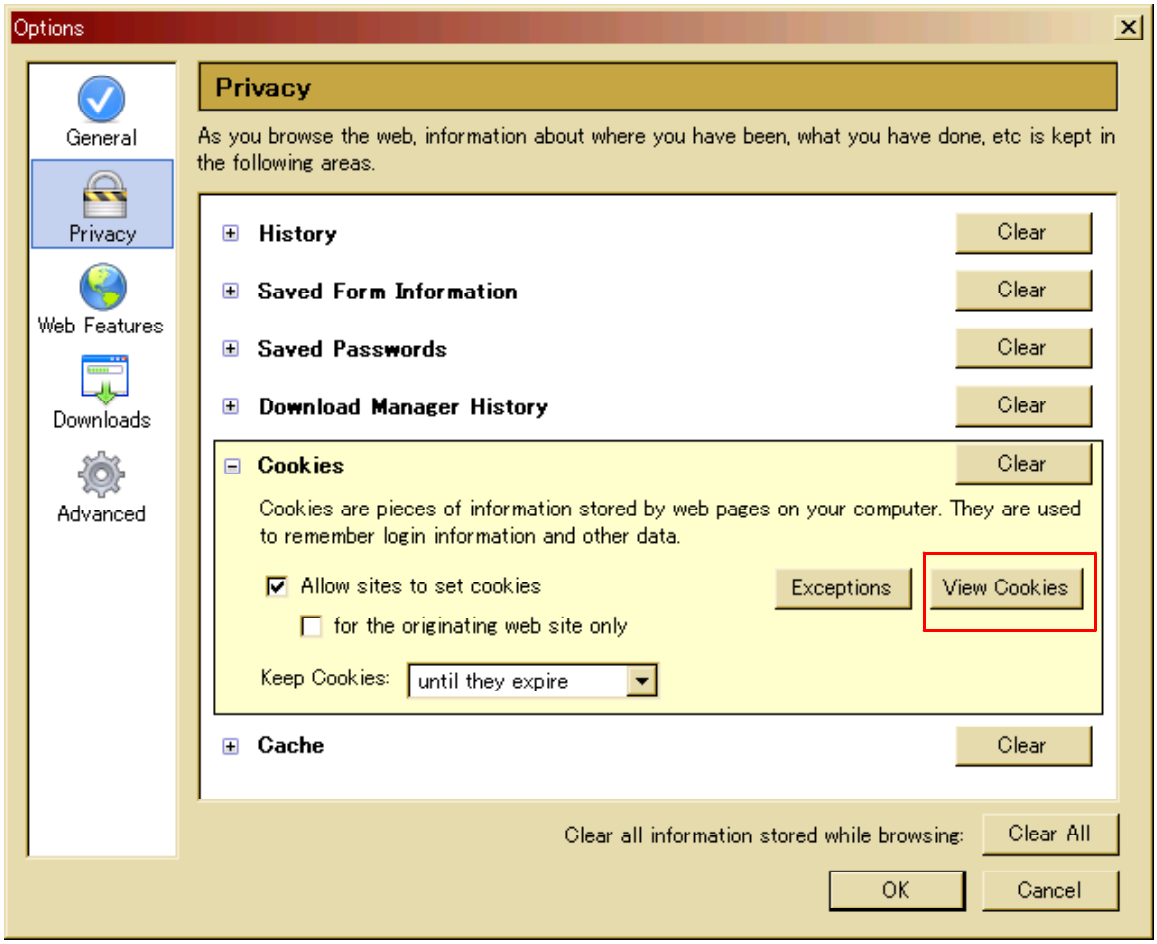


あー・・・何気に末尾の text/html が不穏な内容になってますが（だからPHPスクリプト、違うんですってばてば。勘弁してえな）。
までも、一応

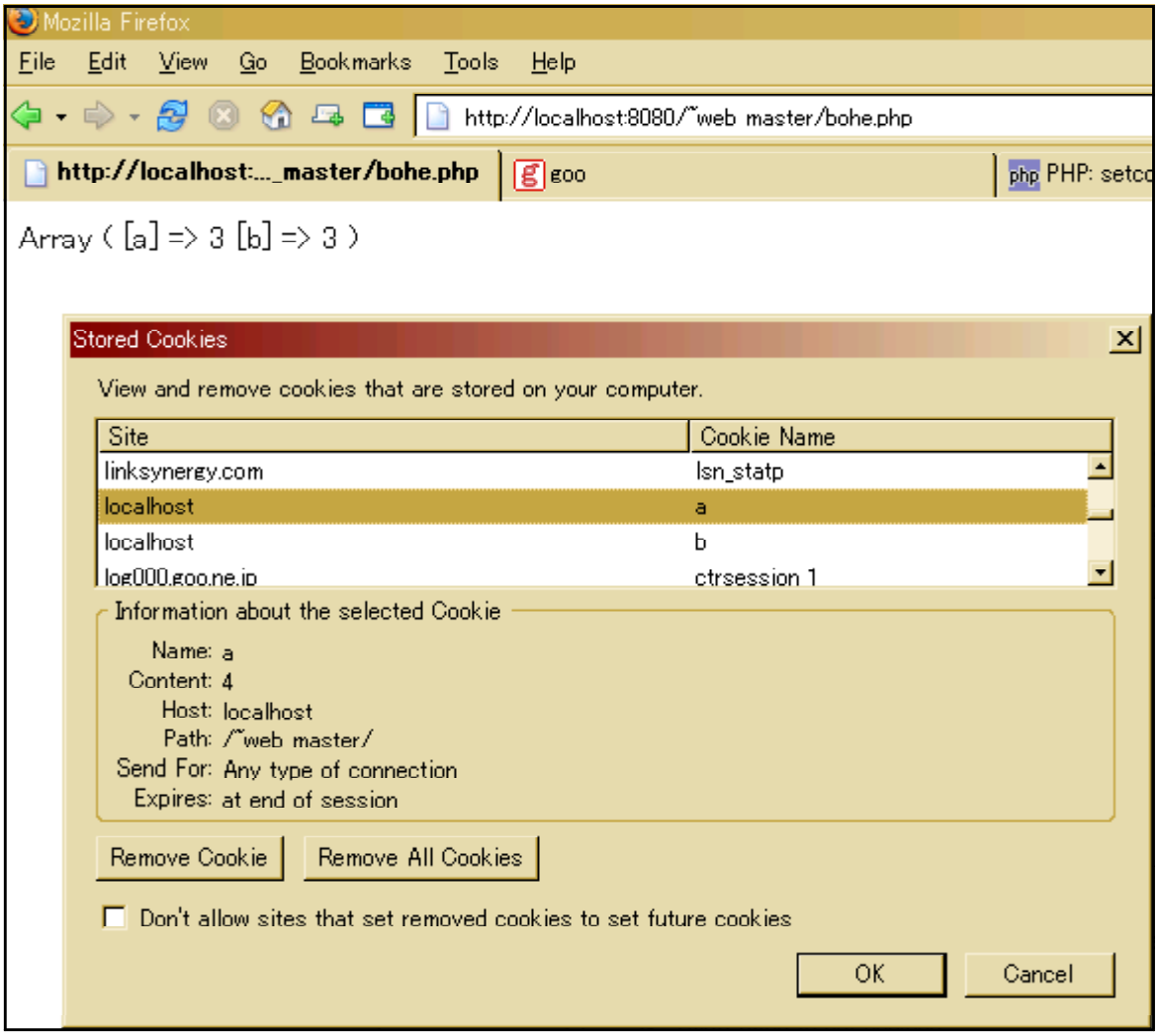
```
...
else setcookie("a", ++$HTTP_COOKIE_VARS["a"]);
...
else setcookie("b", ++$HTTP_COOKIE_VARS["b"]);
```

により、「クッキーはこれこれにしてね。」と Set-Cookie が新しい値で送信されています！

で。結局、このクッキーはブラウザではどのようにして管理されているのでしょうか？
実際に実体としてどう管理されているのではなくて（ブラウザ依存だし）、ブラウザの備えるクッキー管理ではどうなるのか、を見てみます。それくらいだったらどのブラウザも一緒だし。
FireFoxの場合、「Tools」→「Options」→「Privacy」の「+Cookies」を展開して「View Cookies」をクリックします。



すると、こんな具合に。



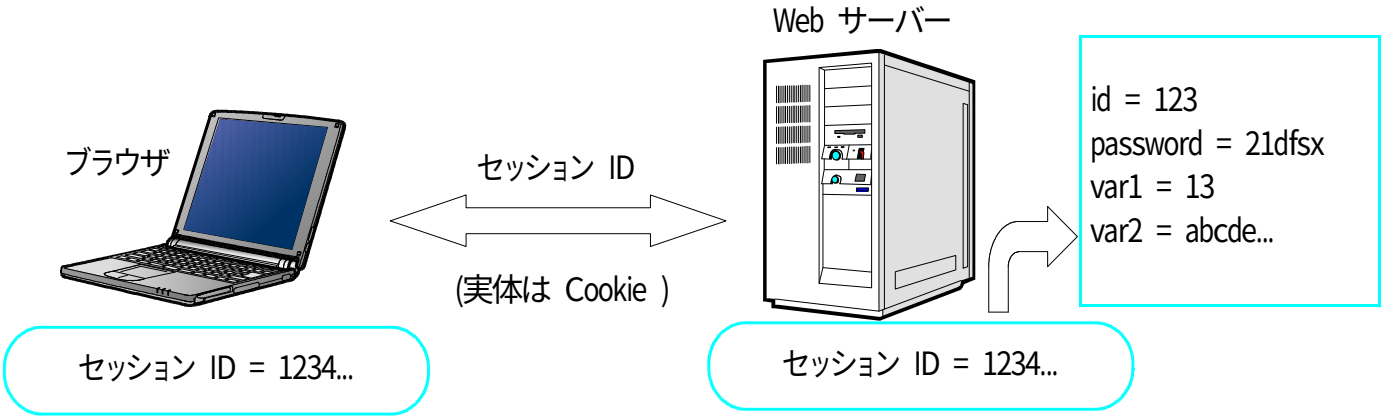
こんな具合に、サイト毎に管理されます。ここで「Remove Cookie」とかすればブラウザ側で保持してたCookieの値が消えますので、次回リクエスト時には「Cookie」ヘッダーが無くなり、鯖からの「Set-Cookie」で改めてクッキーが再設定されるわけです。

いかがでしょう。
こうしてみると、要するにクッキーとはブラウザと鯖との間でHTTPヘッダーを介して値をやりとりするものであり、最大の特徴は「値はブラウザ側で保持され（続け）る」点にあることが御納得頂けるかと思います。

４．再度セッションとは
　んで、ですね。やがてクッキーは「ログインユーザーのユーザーIDやパスワード」を保存するのにも使われ出すわけです。これを使えば、うまい具合に一度入力されたユーザー情報を、ログイン情報としてGETやPOSTなど使わずに「恒久的に」ブラウザと鯖の間でやりとりできるわけですから。
　もちろん、クッキーに制限時間をかけられることがHTTPのRFCにありますので、「タイムアウト」エラーなんかも実現できます。

　ところがです。まずクッキーは基本的に「平文」です。盗聴しようと思えばいくらでもできてしまうわけです。次に、ショッピングカートシステムなどをクッキーを使って実装しようと思うと「一変数につき一Cookieヘッダー」ですからCookie変数があっというまに増殖していってしまうわけです。

　そこでようやく、PHPのセッション概念が登場します。これはまとめて言ってしまえば、「値はまとめて鯖側で保管。それを引き出すためのキー値をブラウザとCookieでやりとりする」という仕組みです。



まあ、通常はブラウザ側でセッションの中身を参照することは無いので。そもそもセッションの中身に基づいてHTMLをはき出すのが鯖側のCGIなりなんなりなので。こんな具合で良いわけです。

それでも「セッションID乗っ取り」はあり得るわけですが。根性出せば、の話ですが。

それはともかく、んでは

５．セッションIDの実況中継

を楽しんでみましょう。

今度のPHPはちゃんと動かした奴です（汗）。

```
<?php
session_start();

$HTTP_SESSION_VARS["hoge"]++;
printf("session_id = %s¥n", session_id());
print "hoge=".$HTTP_SESSION_VARS["hoge"];

?>
```

ま、これもアクセスするたびに（リロードするたび）カウントアップしていく変数を表示するだけのサンプルです。では、行ってみましょう。キャプチャフィルタは前と同じ「tcp port 80」です。

①ブラウザからの最初のリクエスト

```
     4 0.002884   192.168.2.2     192.168.2.10    HTTP   GET /~web_master/hoge.php HTTP/1.1
     5 0.003063   192.168.2.10    192.168.2.2     TCP    http > 2097 [ACK] Seq=1 Ack=452 Win=
     6 0.012097   192.168.2.10    192.168.2.2     HTTP   HTTP/1.1 200 OK (text/html)
     7 0.012203   192.168.2.2     192.168.2.10    TCP    2097 > http [ACK] Seq=452 Ack=563 Wi
     8 0.200216   192.168.2.2     192.168.2.10    HTTP   GET /favicon.ico HTTP/1.1
     9 0.201329   192.168.2.10    192.168.2.2     HTTP   HTTP/1.1 404 Not Found (text/html)
    10 0.201433   192.168.2.2     192.168.2.10    TCP    2097 > http [ACK] Seq=840 Ack=1138 W
    11 2.942847   192.168.2.2     192.168.2.10    HTTP   GET /~web_master/hoge.php HTTP/1.1
    12 2.944913   192.168.2.10    192.168.2.2     HTTP   HTTP/1.1 200 OK (text/html)
    13 2.945065   192.168.2.2     192.168.2.10    TCP    2097 > http [ACK] Seq=1343 Ack=1635
    14 3.038214   192.168.2.2     192.168.2.10    HTTP   GET /favicon.ico HTTP/1.1
    15 3.039016   192.168.2.10    192.168.2.2     HTTP   HTTP/1.1 404 Not Found (text/html)
    16 3.039181   192.168.2.2     192.168.2.10    TCP    2097 > http [ACK] Seq=1731 Ack=2210
    17 3.753213   192.168.2.2     192.168.2.10    HTTP   GET /~web_master/hoge.php HTTP/1.1

⊞ Frame 4 (505 bytes on wire, 505 bytes captured)
⊞ Ethernet II, Src: 00:50:56:e1:da:12, Dst: 00:0c:29:3d:ce:cc
⊞ Internet Protocol, Src Addr: 192.168.2.2 (192.168.2.2), Dst Addr: 192.168.2.10 (192.168.2.10)
⊞ Transmission Control Protocol, Src Port: 2097 (2097), Dst Port: http (80), Seq: 1, Ack: 1, Len: 451
⊟ Hypertext Transfer Protocol
  ⊞ GET /~web_master/hoge.php HTTP/1.1\r\n
    Host: localhost:8080\r\n
    User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; rv:1.7.3) Gecko/20040913 Firefox/0.10.1\r\n
    Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=
    Accept-Language: ja,en-us;q=0.7,en;q=0.3\r\n
    Accept-Encoding: gzip,deflate\r\n
    Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
    Keep-Alive: 300\r\n
    Connection: keep-alive\r\n
    Cache-Control: max-age=0\r\n
    \r\n
```

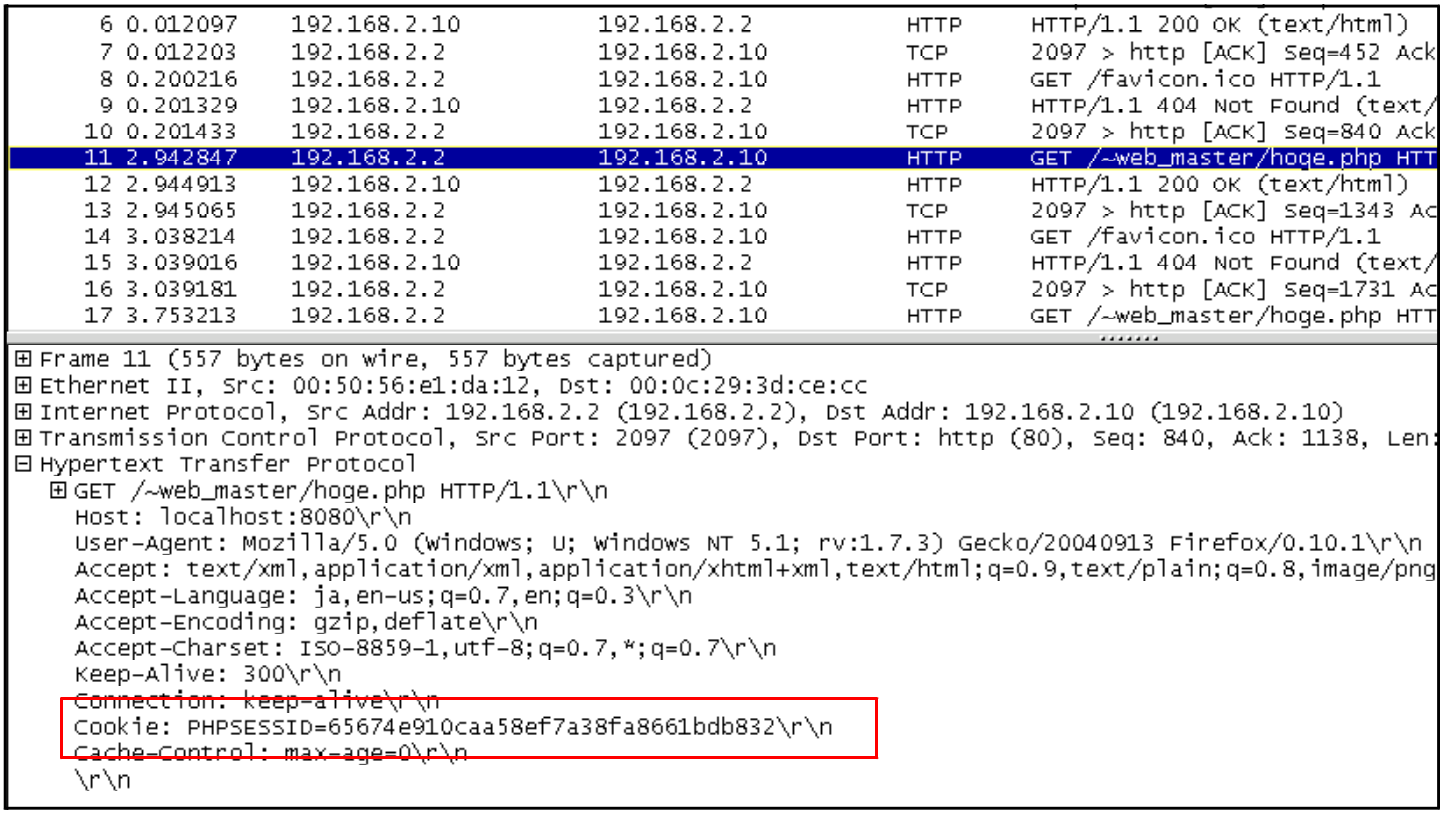
なんてこたない、普通のリクエストです。
んでは。

②鯖応答一回目

```
     5 0.003063   192.168.2.10    192.168.2.2     TCP    http > 2097 [ACK] Seq=1 Ack=45
     6 0.012097   192.168.2.10    192.168.2.2     HTTP   HTTP/1.1 200 OK (text/html)
     7 0.012203   192.168.2.2     192.168.2.10    TCP    2097 > http [ACK] Seq=452 Ack=
     8 0.200216   192.168.2.2     192.168.2.10    HTTP   GET /favicon.ico HTTP/1.1
     9 0.201329   192.168.2.10    192.168.2.2     HTTP   HTTP/1.1 404 Not Found (text/h
    10 0.201433   192.168.2.2     192.168.2.10    TCP    2097 > http [ACK] Seq=840 Ack=
    11 2.942847   192.168.2.2     192.168.2.10    HTTP   GET /~web_master/hoge.php HTTP
    12 2.944913   192.168.2.10    192.168.2.2     HTTP   HTTP/1.1 200 OK (text/html)
    13 2.945065   192.168.2.2     192.168.2.10    TCP    2097 > http [ACK] Seq=1343 Ack
    14 3.038214   192.168.2.2     192.168.2.10    HTTP   GET /favicon.ico HTTP/1.1
    15 3.039016   192.168.2.10    192.168.2.2     HTTP   HTTP/1.1 404 Not Found (text/h
    16 3.039181   192.168.2.2     192.168.2.10    TCP    2097 > http [ACK] Seq=1731 Ack
    17 3.753213   192.168.2.2     192.168.2.10    HTTP   GET /~web_master/hoge.php HTTP

⊞ Frame 6 (616 bytes on wire, 616 bytes captured)
⊞ Ethernet II, Src: 00:0c:29:3d:ce:cc, Dst: 00:50:56:e1:da:12
⊞ Internet Protocol, Src Addr: 192.168.2.10 (192.168.2.10), Dst Addr: 192.168.2.2 (192.168.2.2)
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: 2097 (2097), Seq: 1, Ack: 452, Len: 562
⊟ Hypertext Transfer Protocol
  ⊞ HTTP/1.1 200 OK\r\n
    Date: Mon, 18 Oct 2004 12:05:02 GMT\r\n
    Server: Apache/1.3.27 (Unix) mod_ruby/0.9.7 Ruby/1.6.4 mod_perl/1.26 DAV/1.0.3 PHP/4.2.3\r\n
    X-Powered-By: PHP/4.2.3\r\n
    Set-Cookie: PHPSESSID=65674e910caa58ef7a38fa8661bdb832; path=/\r\n
    Expires: Thu, 19 Nov 1981 08:52:00 GMT\r\n
    Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0\r\n
    Pragma: no-cache\r\n
    Keep-Alive: timeout=15, max=100\r\n
    Connection: Keep-Alive\r\n
    Transfer-Encoding: chunked\r\n
    Content-Type: text/html; charset=euc-jp\r\n
    \r\n
  ⊞ HTTP chunked response
⊟ Line-based text data: text/html
    session_id = 65674e910caa58ef7a38fa8661bdb832
    hoge=1
```

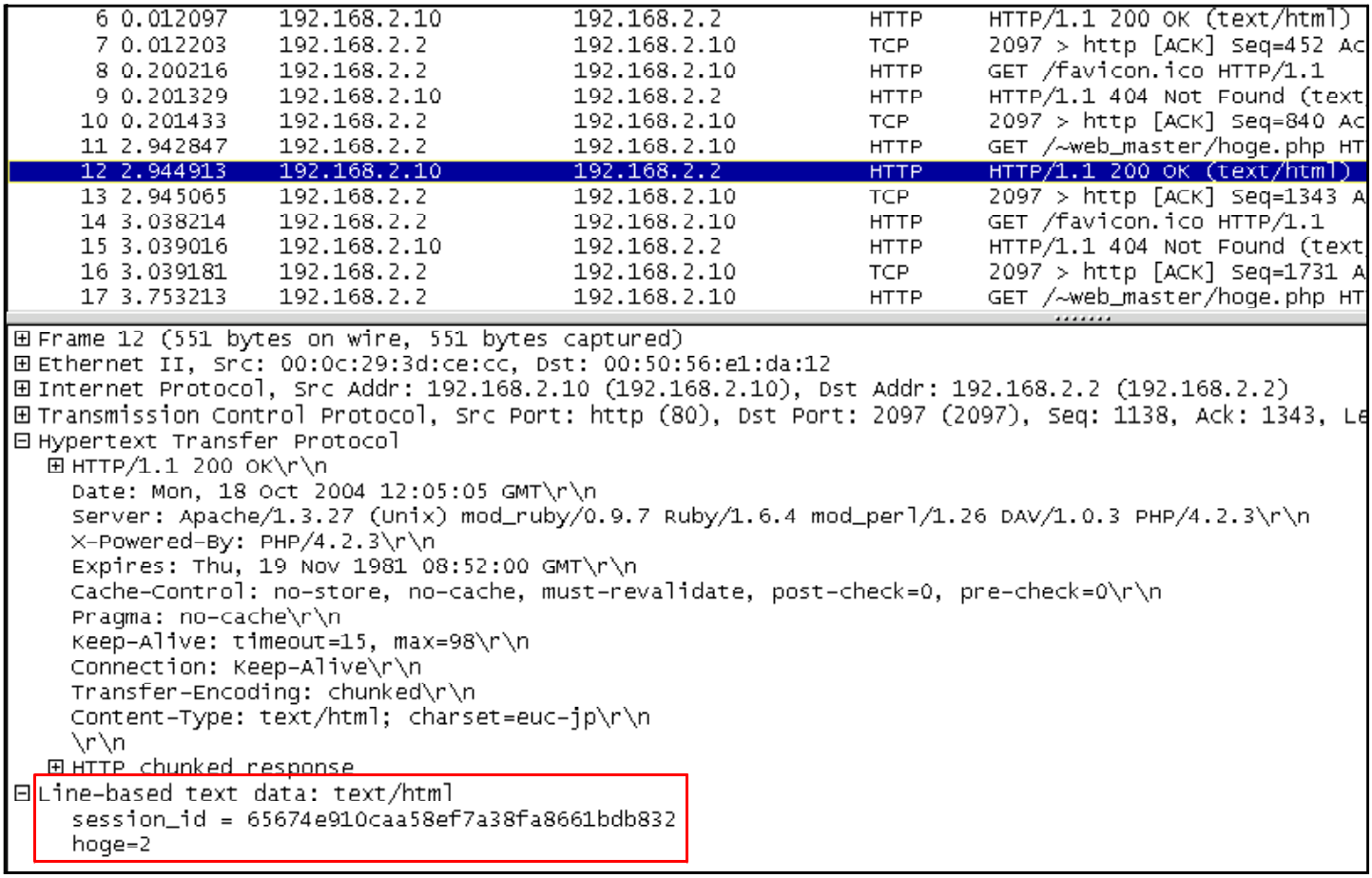
「Set-Cookie」に注目してください。普通の「Set-Cookie」ヘッダーとなんら代わりありません。ただ、変数名が「PHPSESSID」という変数になってます。(続けてpathがあります。これも実はセッションと関連しますが・・・)
で、実際のHTMLボディを表すデータ部を見てみましょう。これは、PHPスクリプトの
printf("session_id = %s¥n", session_id());
print "hoge=".$HTTP_SESSION_VARS["hoge"];
が出力している部分です。
session_id()でセッションIDを取得しているわけです。で、セッションIDすなわちクッキーの値、というわけです。

③二回目のブラウザリクエスト



はい。今までのクッキーと同様に、先ほどのSet-Cookieで送られてきたクッキー値を使ってCookieを送信してます。(path変数は空なので、送信しなかったようです)
中身の「hoge」という値は、ブラウザは関知してないようです。

④鯖応答二度目

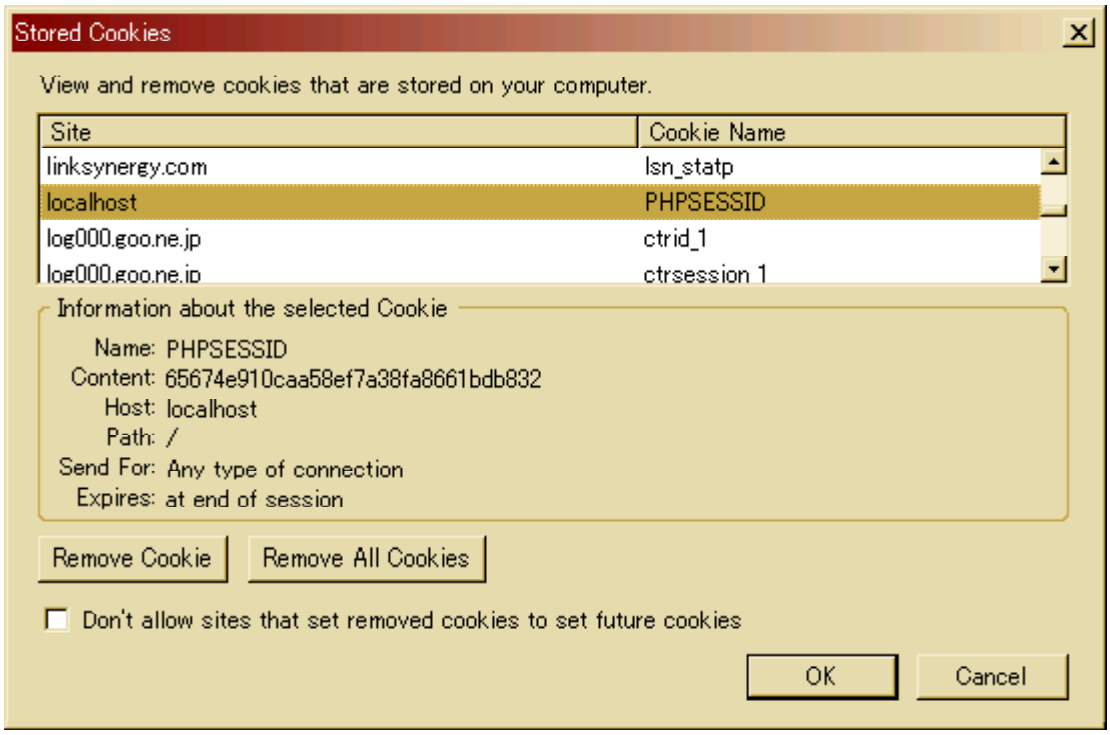


・・・どうでしょうか。「Set-Cookie」ヘッダーが消えてます。
これは単純で、逆に何で前のCookieの実験で毎度Set-Cookieがあったのかを考えればわかりやすいと思います。

何度も言いますが、クッキーの値はブラウザ側で保持します。
Set-Cookieというのは、ブラウザ側に「新しいクッキーの値」を報知するための鯖用（たぶん）のヘッダーです。んで、先ほどの例ではクッキーの値を毎度+1してました。だから、毎度新しい値をブラウザに知らせるため、Set-Cookieを送信していたわけです。

ところがセッションIDというのは、ある一定の期間(session)同じキー値を「ブラウザ側が鯖に報知」することにより、鯖側でブラウザの身元証明代わりに使います。ですから、セッションIDは一度「Set-Cookie」すれば良いわけです。
ブラウザが「Cookie」でセッションIDを送らなくなったとき。それは、ブラウザ側で（セッションID用の）クッキーの保持期限が過ぎたことを意味しますので、「タイムアウトエラー」みたく考えて（多くのアプリでは）ログインフォームの再表示などを行うわけです。

んでは、この段階でブラウザ側ではどんなふうにセッションIDが「クッキーとして」管理されているのでしょうか？



普通のクッキーと変わりません。

こんな具合で、PHPのセッションの実体はクッキーであることが納得頂けたかと・・・って、まだもう一つ重要なの忘れてました。

これ、肝心の「hoge」という変数の値はいったいどこに保存されてるんでしょうか？
ブラウザ側ではなく、鯖側にあるのは分かるのですが・・・

その秘密は、いつもの <?php phpinfo(); ?>を動かせば分かります。

６．セッションの実体（ファイルバージョン）

session

Session Support	enabled

Directive	Local Value	Master Value
session.auto_start	Off	Off
session.cache_expire	180	180
session.cache_limiter	nocache	nocache
session.cookie_domain	*no value*	*no value*
session.cookie_lifetime	0	0
session.cookie_path	/	/
session.cookie_secure	Off	Off
session.entropy_file	*no value*	*no value*
session.entropy_length	0	0
session.gc_maxlifetime	1440	1440
session.gc_probability	1	1
session.name	PHPSESSID	PHPSESSID
session.referer_check	*no value*	*no value*
session.save_handler	files	files
session.save_path	/tmp	/tmp
session.serialize_handler	php	php
session.use_cookies	On	On
session.use_trans_sid	0	0

「session.save_path」という、いかにもそれっぽいのがPHP変数としてリストアップされています。では、早速見てみましょう。

```
[web_master@hogasumi /tmp]$ ls
gpmOLROyz=  gpmMaYQHm=  gpmbE4zsv=  gpmiJfnAX=       orbit-root/                      sess_bcebbcaa0f40131c3c1550a5c5133c0e
gpm2Yx4sF=  gpmNWFwAT=  gpmcsOWZr=  gpmjHvvyC=       sess_2c4a102c680bede7847a153ff93315dd  sess_cc3c729b02b8474d2371458af3985843
gpm8pJ1je=  gpmRJRHLd=  gpmdxjwBR=  gpmmEjkbT=       sess_65674e910caa58ef7a38fa8661bdb832  sess_eb2a3a6eeaa9b5180aa5fbc012e50d0f
gpmB7mOLk=  gpmSRFErG=  gpmfBPFQD=  gpmuUkOqr=       sess_a17e6e187ce3043ad13c17c547d02dcc  vmware-linux-tools.tar.gz
gpmFZPByZ=  gpmVwb1Qg=  gpmhjJImI=  mcop-web_master/  sess_ada53343e1ff36a980056f239710188c
[web_master@hogasumi /tmp]$
```

ありました。一目でわかります。FireFoxのクッキー管理画面やsession_id()の出力で得られた「セッションID」と同じ値で構成された名前です。

```
drwx------  2 root    root      4096 Jul 15 23:47 orbit-root/
-rw-------  1 nobody  nobody       9 Oct 18 21:12 sess_2c4a102c680bede7847a153ff93315dd
-rw-------  1 nobody  nobody       9 Oct 18 21:08 sess_65674e910caa58ef7a38fa8661bdb832
-rw-------  1 nobody  nobody       9 Oct 18 20:33 sess_a17e6e187ce3043ad13c17c547d02dcc
-rw-------  1 nobody  nobody       9 Oct 18 21:10 sess_ada53343e1ff36a980056f239710188c
-rw-------  1 nobody  nobody      52 Oct  8 00:42 sess_bcebbcaa0f40131c3c1550a5c5133c0e
-rw-------  1 nobody  nobody      44 Oct 16 09:45 sess_cc3c729b02b8474d2371458af3985843
-rw-------  1 nobody  nobody       9 Oct 18 21:01 sess_eb2a3a6eeaa9b5180aa5fbc012e50d0f
```

ユーザーもnobody。間違いありません。中身を見てみましょう。

```
[root@hogasumi /tmp]# more sess_65674e910caa58ef7a38fa8661bdb832
hoge|i:4;
[root@hogasumi /tmp]# more sess_2c4a102c680bede7847a153ff93315dd
hoge|i:4;
```

ようやく見えてきました。

変数名|変数の型:値;

で一つの変数を表しているようです。

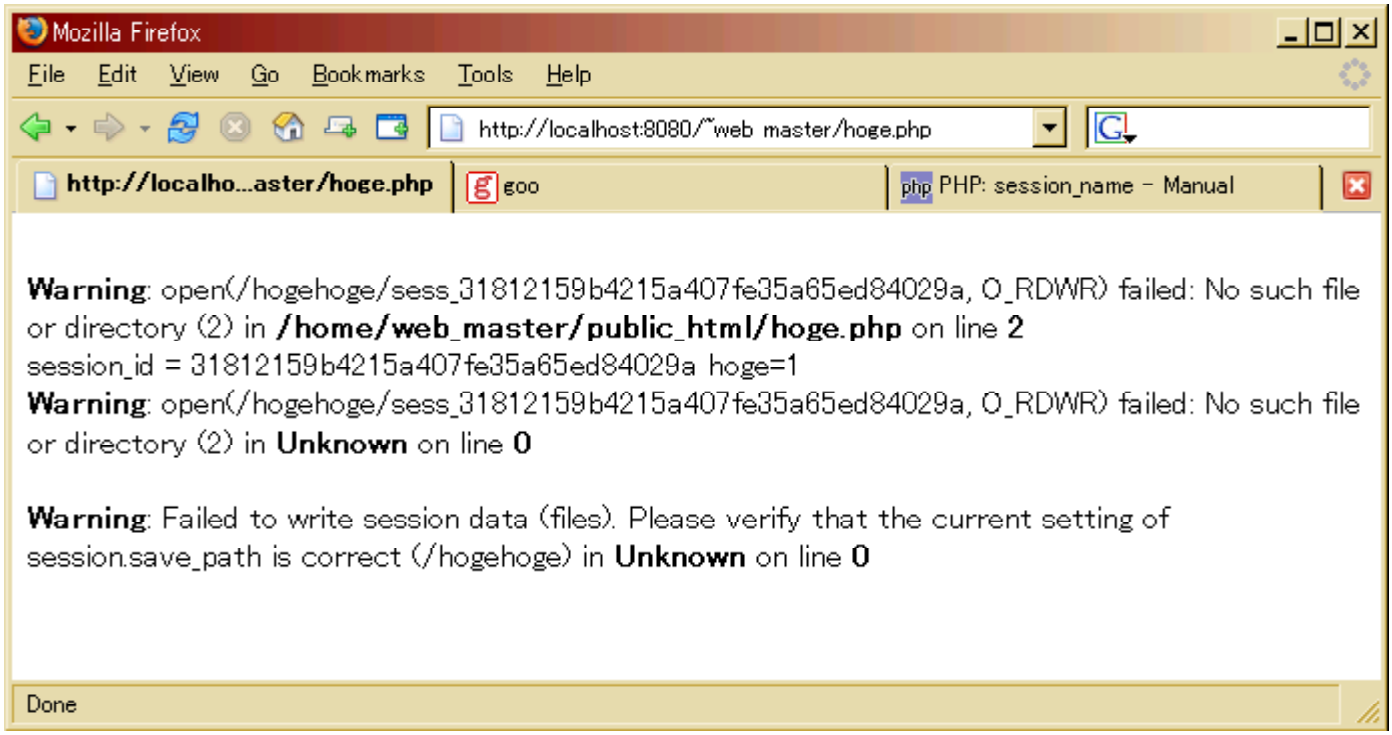
ではここでちょっと遊んでみます。save_pathを変更して存在しないディレクトリにし、セッションファイルが作成できないようにしてみましょう。PHPスクリプトと同じディレクトリに .htaccess をおき、中身を以下のようにしてみます。

```
<IfModule mod_php4.c>
        php_flag output_buffering on
        php_value session.save_path /hogehoge
</IfModule>
```

もちろん、/hogehoge は存在しないディレクトリです。
これでphpinfoの出力は・・・

session.save_path	/hogehoge	/tmp

大丈夫そうです。左がローカル設定、右が全体の基本設定です。
では、アクセスしてみます。



見事！予想通り、いかにも「セッションファイルが作れないからセッションが取り出せないよう」って具合のワーニングメッセージです。

実は、Windows用のPHPパッケージのデフォルトのphp.iniでは、このsession.save_pathが正しく設定されていません。（まあ、Windowsではテンポラリフォルダの位置がUnixほど定型的ではないので・・・とか。）したがって、Windows上でPHPのセッション機構を利用す

るときは、まずはphp.iniのsession.save_pathでフォルダを設定する必要があることをよく注意していください。

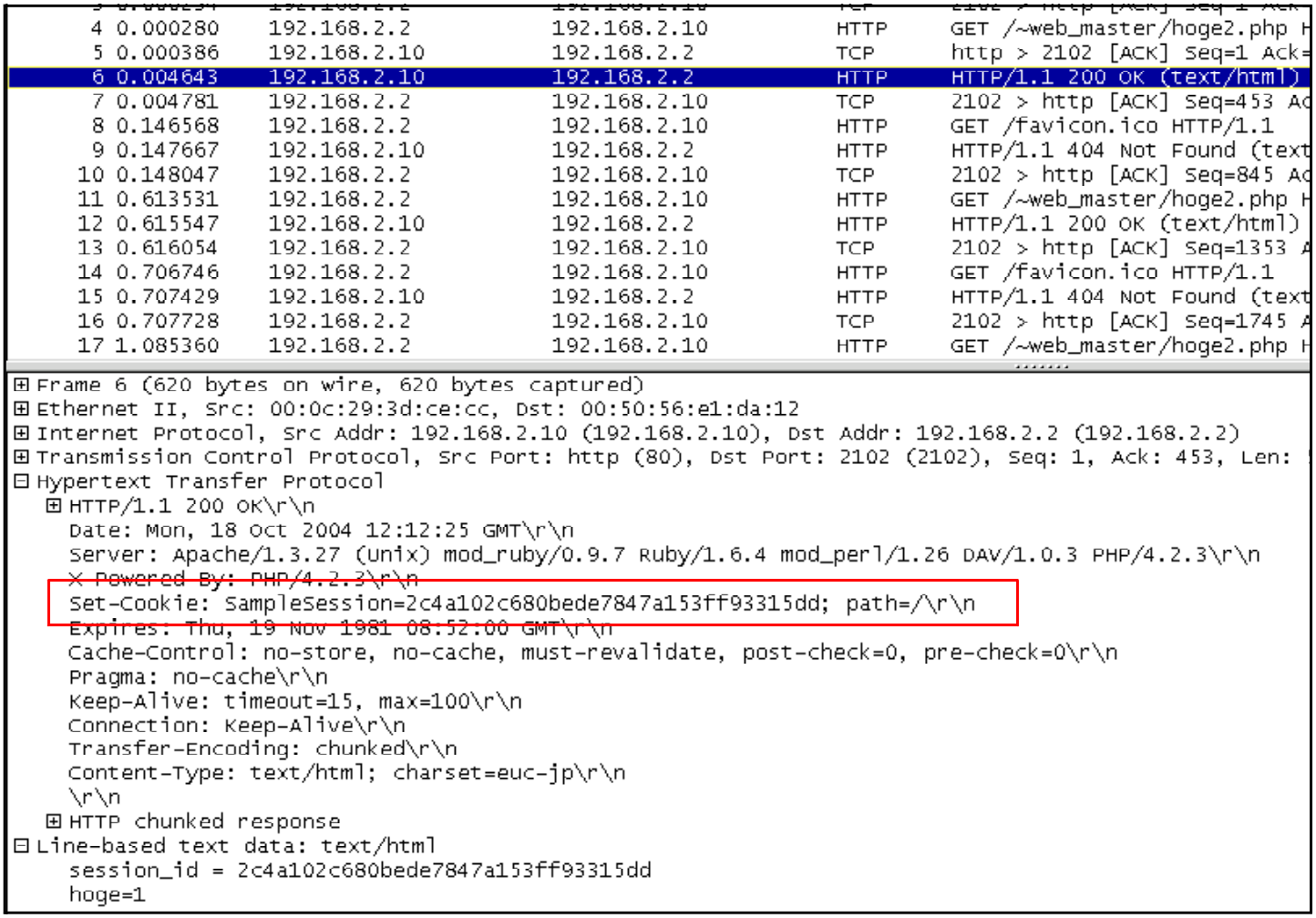
７．セッションIDのクッキー変数名を変更する

さて。セッションIDが保存されるクッキー変数名、PHPSESSIDっていかにも味気ないです。
同じ鯖上でことなるシステムが使うことを想定すると、PHPSESSIDという変数名、いかにも変更できるっぽいです。

んで、実際に変更できます。session_name()で設定できます。

```
<?php
session_name("SampleSession");
session_start();

$HTTP_SESSION_VARS["hoge"]++;
printf("session_id = %s¥n", session_id());
print "hoge=".$HTTP_SESSION_VARS["hoge"];

?>
```

①鯖の初回 Set-Cookie



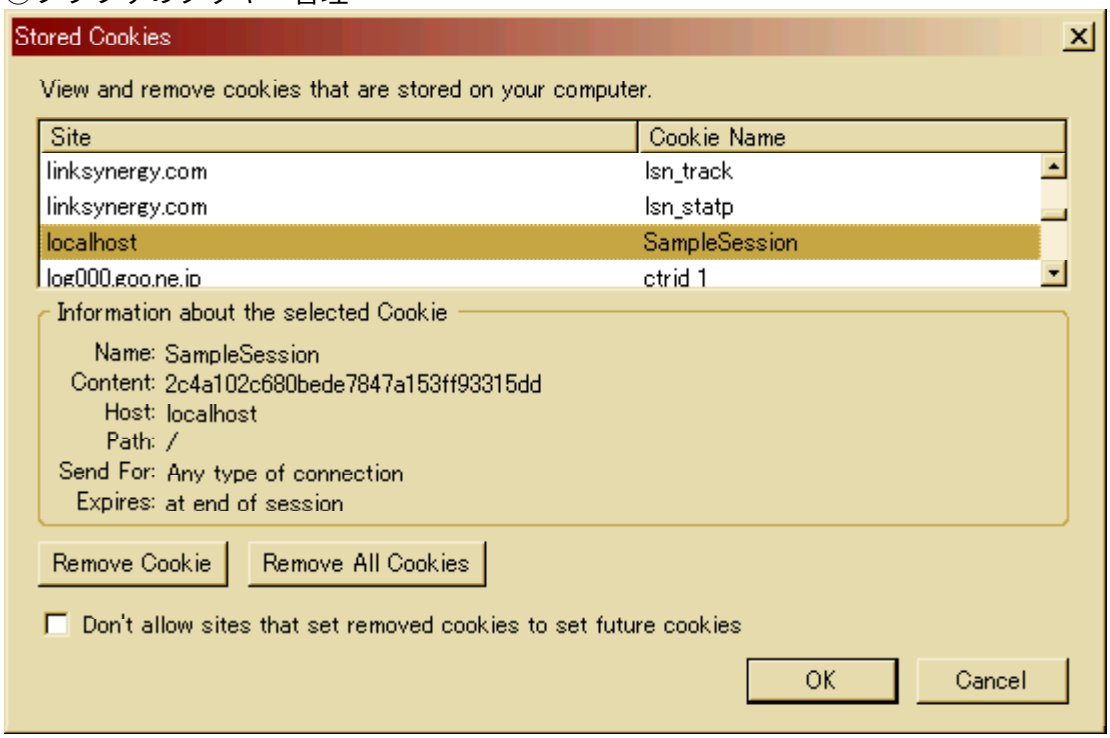
見事にセッションクッキーの変数名が変更されています。

②ブラウザのリクエスト(Set-Cookie後)

```
     4 0.000280    192.168.2.2     192.168.2.10    HTTP    GET /~web_master/hoge2.php HTTP/1.1
     5 0.000386    192.168.2.10    192.168.2.2     TCP     http > 2102 [ACK] Seq=1 Ack=453 Win=
     6 0.004643    192.168.2.10    192.168.2.2     HTTP    HTTP/1.1 200 OK (text/html)
     7 0.004781    192.168.2.2     192.168.2.10    TCP     2102 > http [ACK] Seq=453 Ack=567 Wi
     8 0.146568    192.168.2.2     192.168.2.10    HTTP    GET /favicon.ico HTTP/1.1
     9 0.147667    192.168.2.10    192.168.2.2     HTTP    HTTP/1.1 404 Not Found (text/html)
    10 0.148047    192.168.2.2     192.168.2.10    TCP     2102 > http [ACK] Seq=845 Ack=1142 W
    11 0.613531    192.168.2.2     192.168.2.10    HTTP    GET /~web_master/hoge2.php HTTP/1.1
    12 0.615547    192.168.2.10    192.168.2.2     HTTP    HTTP/1.1 200 OK (text/html)
    13 0.616054    192.168.2.2     192.168.2.10    TCP     2102 > http [ACK] Seq=1353 Ack=1639 W
    14 0.706746    192.168.2.2     192.168.2.10    HTTP    GET /favicon.ico HTTP/1.1
    15 0.707429    192.168.2.10    192.168.2.2     HTTP    HTTP/1.1 404 Not Found (text/html)
    16 0.707728    192.168.2.2     192.168.2.10    TCP     2102 > http [ACK] Seq=1745 Ack=2214 W
    17 1.085360    192.168.2.2     192.168.2.10    HTTP    GET /~web_master/hoge2.php HTTP/1.1
⊞ Frame 11 (562 bytes on wire, 562 bytes captured)
⊞ Ethernet II, Src: 00:50:56:e1:da:12, Dst: 00:0c:29:3d:ce:cc
⊞ Internet Protocol, Src Addr: 192.168.2.2 (192.168.2.2), Dst Addr: 192.168.2.10 (192.168.2.10)
⊞ Transmission Control Protocol, Src Port: 2102 (2102), Dst Port: http (80), Seq: 845, Ack: 1142, Len: 508
⊟ Hypertext Transfer Protocol
  ⊞ GET /~web_master/hoge2.php HTTP/1.1\r\n
    Host: localhost:8080\r\n
    User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; rv:1.7.3) Gecko/20040913 Firefox/0.10.1\r\n
    Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=
    Accept-Language: ja,en-us;q=0.7,en;q=0.3\r\n
    Accept-Encoding: gzip,deflate\r\n
    Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
    Keep-Alive: 300\r\n
    Connection: keep-alive\r\n
    Cookie: SampleSession=2c4a102c680bede7847a153ff93315dd\r\n
    Cache-Control: max-age=0\r\n
    \r\n
```

ちゃんとセッションクッキー変数名が取れているようです。

③ブラウザのクッキー管理



大丈夫そうです。

８．HTML出力バッファリング(output_buffering, ob_start())

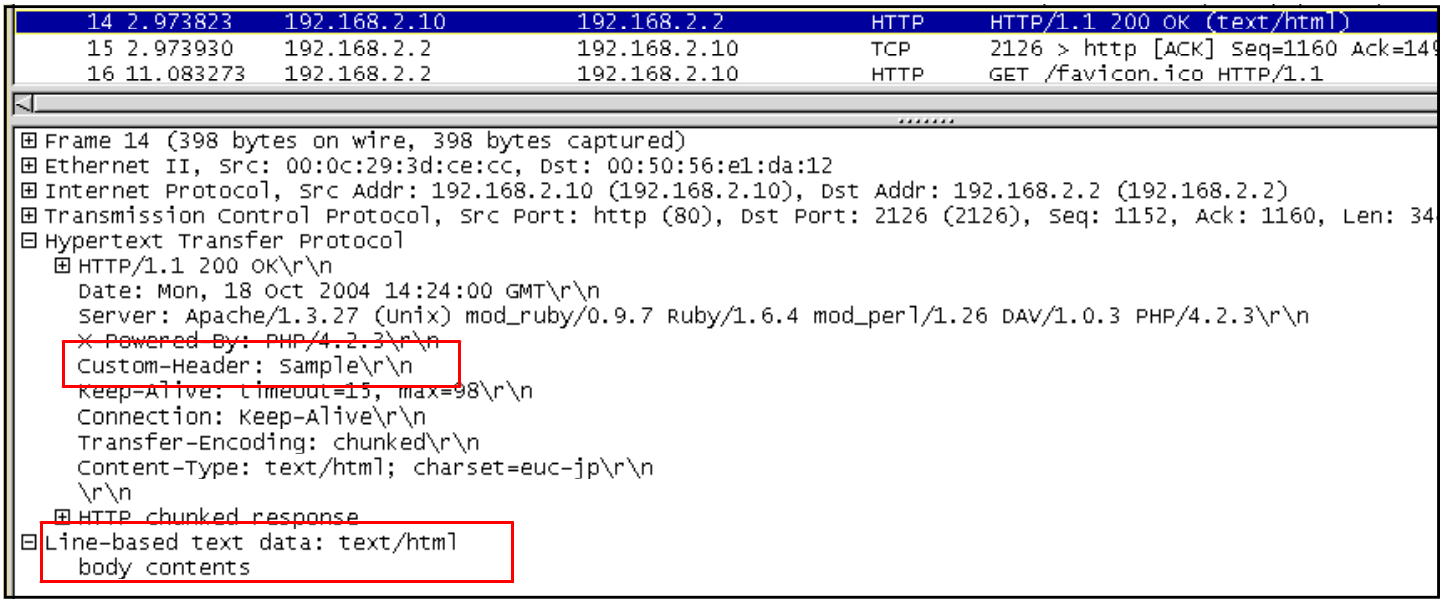
さて、PHPでは header関数や setcookie, session_start などでHTTPヘッダーの要素を出力できます。
できるのですが・・・もしも。もしも、途中でエラーなどが起こって、HTTPヘッダーを出力する「前に」無関係なエラー文字列や、HTMLコンテンツが出力されたらどうなるのでしょうか？

ちょっとやってみましょう。

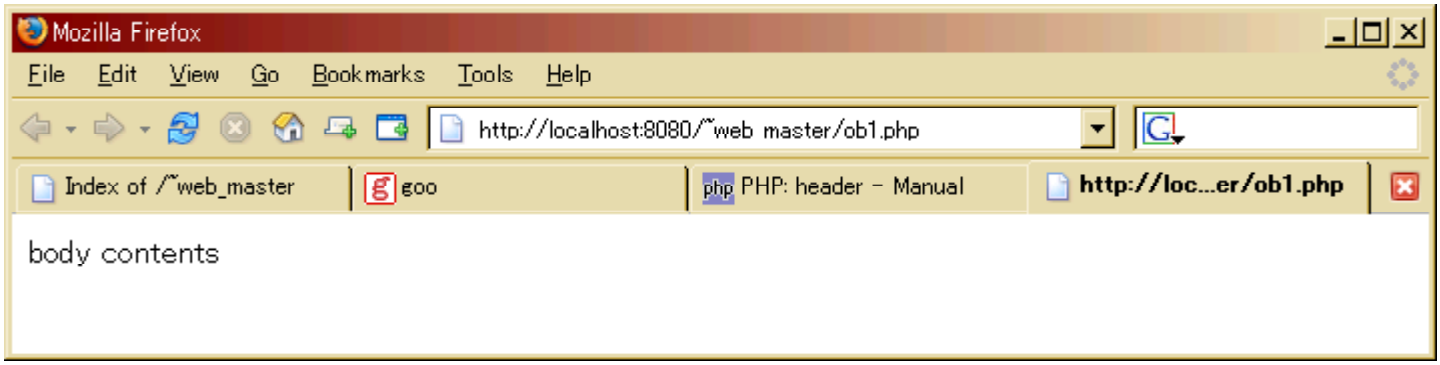
まずは、正常系です。

```
<?php
header("Custom-Header: Sample");
print "body contents";
?>
```

これにアクセスしたときの、鯖のレスポンスを下に示します。



Webブラウザ出力を下に示します。

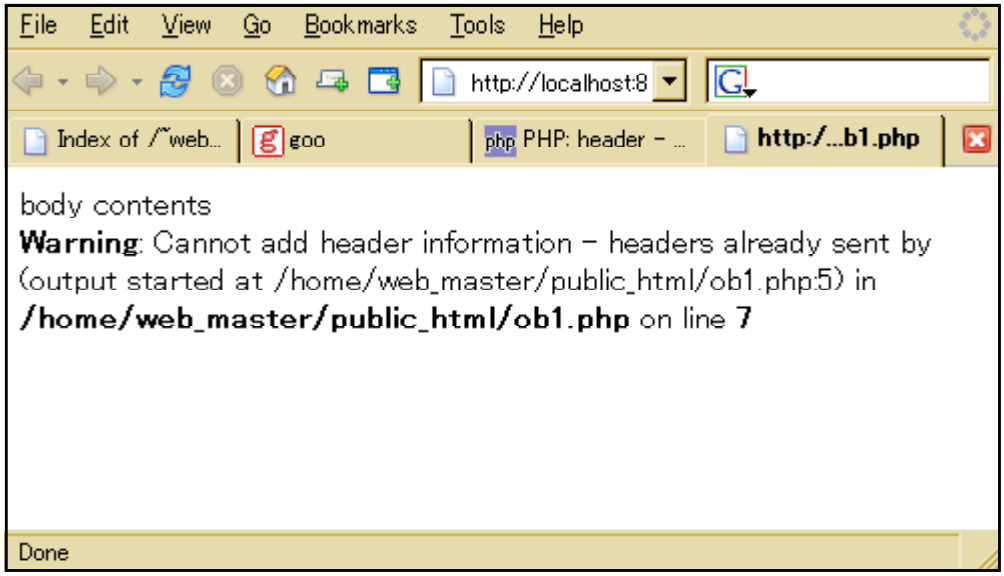


まずはスクリプト通り、「Custom-Header: Sample」というHTTPヘッダーが出力され、次に通常のHTMLボディコンテンツに相当する"body contents"文字列が出力され、ブラウザ画面にも表示されています。

では、順番を入れ替えてみます。

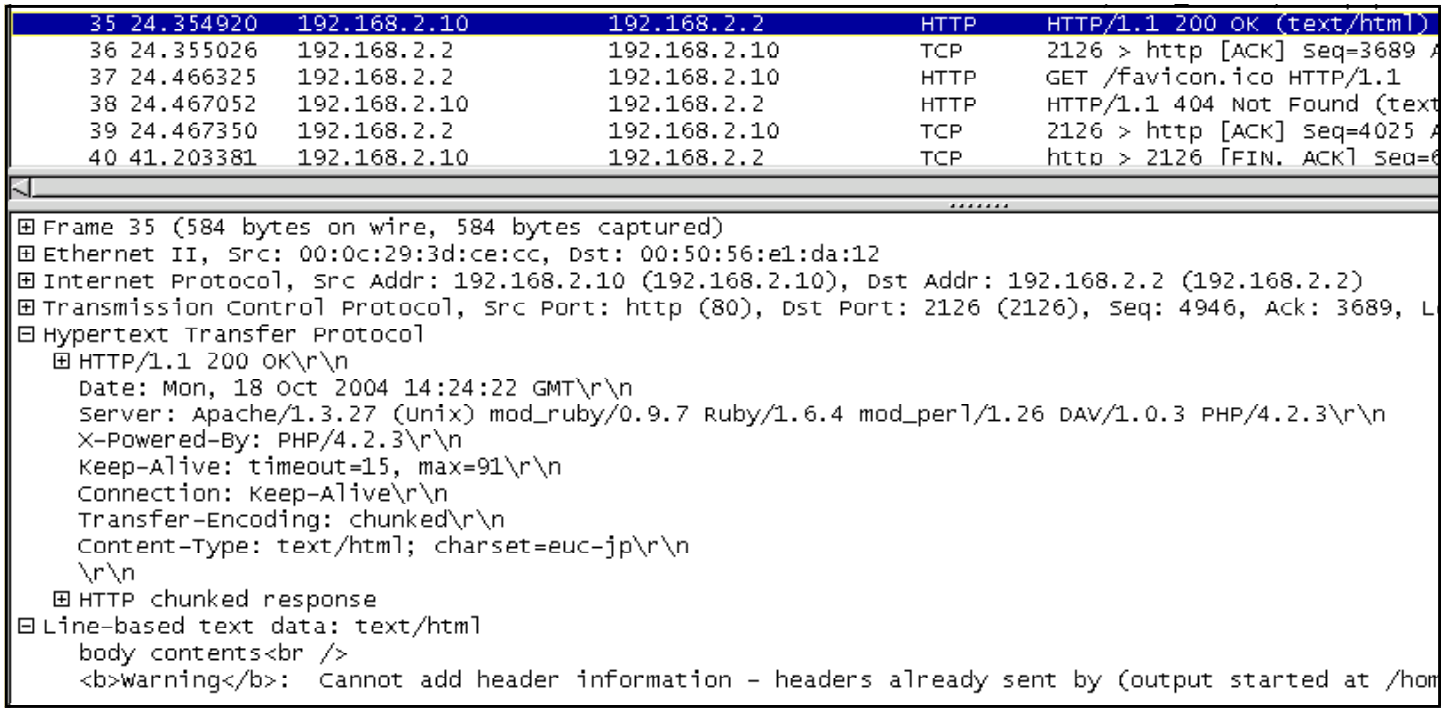
```
<?php
print "body contents";
header("Custom-Header: Sample");
?>
```

こうすると・・・今度は最初にWeb画面をば。



はい。PHP側で、「ヲイヲイ、HTMLコンテンツを送信した後にHTTPヘッダー送信されても困るんだよ、チミ。」ってな具合の警告が表示されちゃいます。

実際のHTTPパケットはどうなったのでしょうか？



・・・「Custom-Header」など影も形も無くなりました。

このように、ヘッダー出力関連（代表的な関数だと header, setcookie, session_start, session_name）のPHP関数は、呼び出す前に通常出力が行われてはなりません。

が。

よーくこの一文を読んでみてください。「出力が」行われてはならないのです。なぜわざわざ「出力処理」（つまりエラーメッセージ出力とか print 系諸々）と書かずに、「出力」と書くのか。

これが、「出力バッファリング」という裏技（いえ、実際は裏でも何でも無いですが。まあ初心者向けの解説で

はまず出てこない・・・ん？最近の解説は質がよいから、出てくるかも？）があるわけです。

「出力バッファリング」とは、print 系やエラーメッセージ出力系の、「出力処理」によって出力された文字列をPHPの内部バッファにためておき、HTTPヘッダーが全部出力し終わってから実際にパケットとして送信してくれる便利な機能です。

mb_string系の文字コード変換機能も、この「出力バッファリング」を利用してます。

実際にどう使うのか？二種類、方法があります。

①ob_start()関数を使う。

```
<?php

ob_start();

print "body contents";
header("Custom-Header: Sample");
?>
```

ob_start()が実行された時点で出力バッファリングが始まります。出力バッファをクリアしたいときは、ob_flush()を使うそうです（未確認)。というわけで、使い方によってはすんごい便利そう。

②output_buffering PHP環境変数を "on" にセットする。

php.iniを書き換える手もありますが、結構影響範囲が大きいと思います。そういうときは、先ほども登場した .htaccess ファイルを用いた局所化を行います。

```
<IfModule mod_php4.c>
        php_flag output_buffering on
</IfModule>
```

これで、.htaccess と同じディレクトリにあるPHPスクリプト全部で出力バッファリングが有効化されます。

このいずれかの方法で、間違いなく、Warning は消えて「Custom-Header」も復活するはずです。

９．載せきれなかったヒントとか、おまけとか。

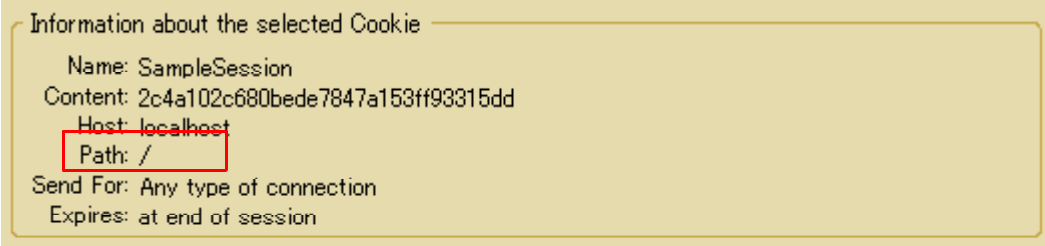
さて、同じセッション認証ライブラリを用いているシステムが同じドメイン内で動いていたりするとき、下手すると、「同じ鯖で同じセッションクッキー変数名を使ってしまう」みたいになりかねません。
たとえば、http://www.xxxx.com/webapp1/ というPHPアプリケーションと、http://www.xxx.com/webapp2/ というPHPアプリケーションとではセッションクッキー変数名を分ける必要が出てくるわけです。

そのための識別要素が、「path」要素です。

・鯖レスポンスより

```
X-Powered-By: PHP/4.2.3\r\n
Set-Cookie: PHPSESSID=65674e910caa58ef7a38fa8661bdb832; path=/\r\n
Expires: Thu, 19 Nov 1981 08:52:00 GMT\r\n
```

・ブラウザ画面より



・http://jp2.php.net/manual/ja/function.session-set-cookie-params.phpより

session_set_cookie_params

(PHP 4 , PHP 5)

session_set_cookie_params -- セッションクッキーパラメータを設定する

説明

void **session_set_cookie_params** (int lifetime [, string path [, string domain [, bool secure]]])

pathをうまく使いこなせば、セッションの識別に相当幅が出てくることでしょう。

これまで見てきたとおりセッションに関してはPHPの文法知識だけでなく、HTTP(Hyper Text Transport Protocol)についても知っておく必要があります。PHPだけではなく、Webアプリケーション開発にはHTTPやTCP/IPの知識が欠かせません。何かトラブった時に、これらの知識やネットワーク関連のツール（特にパケットキャプチャソフト）があると非常に助かると思います。

（Webアプリ開発者にとって）ネットワーク関連は勉強しておいて損になることはありません！

最後に、クッキーやセッション関連の解説サイトを紹介しておきます。
http://www.rfs.jp/sitebuilder/perl/03/04.html
・・・Cookieの基本的な仕様が日本語で掲載されています。
http://jp2.php.net/manual/ja/ref.session.php
・・・PHPのセッション管理の概要が書かれています。まず、これを読むことをおすすめします。
http://jp2.php.net/manual/ja/ref.http.php
・・・クッキーやHTTPヘッダー関連の関数リファレンスです。
http://www.w3.org/Protocols/rfc2616/rfc2616
・・・header関数のリファレンスから飛べます。HTTP/1.1(最新版)のRFC。HTTPの全てはここに（多分）詰まってる。

１０．忘れてた、も一つちょー重要なおまけ。

デフォルトの（敢えて「デフォルト」と付けるところがミソ）セッション保存形式には、実はいろいろと困る点があります。
第一に、何よりも session.save_path から容易に保存ディレクトリが取得でき、しかもしかも、平文なので容易く他のセッションデータがみれてしまうこと。
第二に、セッションの有効期限と関連して、ファイルシステムのタイムスタンプにまつわる話。というのは、ファイルシステムによっては「最終アクセス時刻」がまともに取得できないものもあり、有効期限の厳密な運用に差し障りが出てくるからです。

どうしてもデフォルトのファイルベースでは、平文でしかも設定値から容易くファイルの保管場所が知れてしまうと言うこまった点があります。(sesison_save_path()で変更可能であるし、session.save_pathで特殊な指示方法: '5;/tmp' 等を用いることによりある程度階層化できるとはいえ。)

そこで出てくるのが、session_set_save_handler という関数。

これ、セッション情報の書き出しに使用する関数を自前の関数に切り替えられます。
というわけで、世に使われている有名どころなCMSやらセッション管理PHPライブラリやらDBライブラリでは、みなこぞってこの関数を利用してセッション情報をDBに格納するよう調整しているわけです。

サンプルがぼろぼろ
http://jp2.php.net/manual/ja/function.session-set-save-handler.php
に掲載されていますし、有名どころのCMSやDB管理用ライブラリでは普通に使われてますので、是非一度そこらあたりをHackしてみてください。あるいは上記サンプルを参考に自前で実装してみて、いろいろ遊ぶのもおもしろいと思います。

ってなぐあいで、以上です。おつきあい頂き、誠に有り難うございました。