Security of HTTPHeader

2007 年 10 月 30 日

Ver. 1.2

目　次

1　WebAppli で使う HTTP ヘッダ作成関数の安全な使い方

1.1 本文書の目的

本文書は、Web アプリケーションで利用されている各種 HTTP ヘッダ作成用関数の安全な使い方について検討する。
本文書で対象としている脅威は、主に以下である。

- HTTP ヘッダ・インジェクション (HTTP Response Splitting[※])

HTTP ヘッダ・インジェクションというより、「HTTP Response-Splitting 攻撃」の方が有名であろう。
しかしながら、どの関数が改行コードを許しているのかを調査している文献が見受けられなかった。一方で、[システム開発者 | プログラマ]にとっては、このような情報が必要な情報であると感じ、本文書を作成した。

[※] 「HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics」
https://www.watchfire.com/securearea/whitepapers.aspx?id=8

2　HTTP ヘッダ・インジェクションの概要

2.1 HTTP ヘッダ・インジェクションの概要

Web で使用している HTTP は、テキスト形式の通信プロトコルであり、空行を挟み、前段を「HTTP ヘッダ」、後段を「HTTP ボディ」と分ける構造となっている。
また「HTTP ヘッダ」は、ヘッダ情報を一行に一つを基本形式としている。

Web ブラウザと Web アプリケーション間でのクッキー情報の交換や、Web ページのリダイレクト先を示す URI の情報交換などに HTTP ヘッダは用いられている。

そして、ほとんどの Web アプリケーションには、HTTP ヘッダを操作する関数が用意されている。
Web アプリケーションプログラマが一般的に利用するのは以下の 3 つであると思われる。

- クッキーを Web ブラウザへ送信する関数
- Web ページのリダイレクトを命令する関数
- 任意の HTTP ヘッダを[追加 | 上書]する関数

これらの関数に汚染データ[※]を渡した場合、汚染データ中に改行コード(ヘッダ情報の区切りが改行コードであるため)を挿入することによって、任意の HTTP ヘッダを挿入されてしまうというのが、HTTP ヘッダ・インジェクションである。

また、クッキー情報の各プロパティは「;(セミコロン)」をデリミタとしているため、「;」をはさんで、任意のプロパティを指定される危険性もある。

脅威は、

- 任意の HTTP ヘッダを挿入される
- クッキー情報に任意のプロパティを挿入される
- HTTP Response Splitting

などであるが、現実的な深刻度はそれほど高くない。
しかし、Web アプリケーションを開発する[システム開発者 | プログラマ]の方々には、このようなそれほど深刻度が高くないセキュリティ問題といえども放置せずに、適切なプログラミングを行うことを希望する。

[※] 汚染データ
ここでは、利用者から Web アプリケーションへ入力された安全な書式であると検証されていないデータと定義する。「汚染されているかも知れないデータ」とした方がより正確であるような気がする。

2.2 HTTP ヘッダ・インジェクションへの対策

HTTP ヘッダを操作する関数に汚染データを与える場合、改行コードについてサニタイズ処理を実施する。

ASP/ASP.NET や PHP では、URL エンコードを実施している。
もし、読者が利用している Web アプリケーション用の開発環境の HTTP ヘッダ作成関数内部でサニタイズ処理を行っていないのであれば、ASP/ASP.NET や PHP **の方法を模倣して、**Web **アプリケーションプログラマがそれらの関数を用いる前に、**URL **エンコードを実施すればよいだろう。**

また、クッキー情報の各プロパティは「;(**セミコロン)」をデリミタとしている。利用する関数内部でサニタイズ処理していなければ、**Web **アプリケーションプログラマ自身が事前に「**;(**セミコロン)」を** URL **エンコードすればよいだろう。**

以下の章で明らかとなるが、ASP/PHP **と** Java **とでは、異なるサニタイズ処理がおこなわれていた。**
どちらが正しいのだろうか。

3　HTTP ヘッダ・インジェクションの具体的な安全策

3.1 CGI の場合

CGI では、HTTP ヘッダも含めて HTTP リクエスト全体(HTTP ヘッダ+HTTP ボディ)を CGI プログラマが作成する必要がある。
よって、HTTP ヘッダを作成中に汚染データを使う場合、汚染データ中の改行コードに注意する必要がある。
また、クッキー情報の各プロパティは、「;(セミコロン)」をデリミタとしているため、クッキー情報の HTTP ヘッダを作成する際は「;(セミコロン)」もサニタイズ処理する必要がある。

3.2 Perl + CGI.pm の場合

Perl には、標準で CGI.pm というモジュールがある。このモジュールにある CGI.header()関数、CGI.redirect()関数、CGI::Cookie オブジェクトなどを使う場合も多いのではないだろうか。

CGI.pm での HTTP ヘッダ操作は、以下の関数がよく使われる。

関数名	操作内容
CGI.header()	HTTP ヘッダ生成
CGI.redirect()	Web ページのリダイレクト要求
CGI::Cookie	クッキー情報の作成

これらの関数に汚染データが配置されるようなテストプログラム()を作成し、Netcat を使ってどのような HTTP レスポンスが返るかを観察した。

3．2.1 Perl5.6.1/CGI.pm2.752 + PlamoLinux3.0 の場合

この章では、PlamoLinux3.0(Linux2.4.19) + Perl5.6.1/CGI.pm 2.752 を実験環境として使用した結果である。

図 3．2.1-1～図 3．2.1-2では、CGI.Cookie オブジェクトでクッキーオブジェクトを作り、それをそのまま出力する場合である。

Cookie の値として含まれている改行(Lf)、クッキー属性のデリミタである「;(セミコロン)」が URL エンコード処理が行なわれている。
このようなコードを Perl プログラマが書く場合、「改行」や「;(セミコロン)」を特に意識せずに記述しても安全であるということである。

```
#! /usr/local/bin/perl
use strict;
use CGI;
use CGI::Cookie;

my ${cgi} = new CGI;
my ${str} = "test;¥nnewHeader:value";

my ${myC} = new CGI::Cookie(
        -name=>'cookieName',
        -value=>${str}          );

print "Set-Cookie: ${myC}¥n";
printf("Content-Type: text/html¥n¥n");
printf("${str}¥n");
printf(${cgi}->version());
printf("¥n");
__END__
```

図3．2.1-1：Cookie をそのまま出力するテストスクリプト

```
# ./cgimod-test.pl
Set-Cookie: cookieName=test%3B%0AnewHeader%3Avalue; path=/
Content-Type: text/html

test;
newHeader:value
2.752

# echo "HEAD /cgi-bin/cgimod-test.pl HTTP/1.0¥n¥n" | nc -nvv 127.0.0.1 80
(UNKNOWN) [127.0.0.1] 80 (?) open
```

```
HTTP/1.1 200 OK
Date: Fri, 30 Mar 2007 06:27:19 GMT
Server: Apache/2.0.47 (Unix) mod_ssl/2.0.47 OpenSSL/0.9.6f DAV/2
Set-Cookie: cookieName=test%3B%0AnewHeader%3Avalue; path=/
Connection: close
Content-Type: text/html

sent 40, rcvd 226
```

図3．2.1-2：図 3．2.1-1の実行結果。直接実行しても Apache 経由でも、URL エンコードが行なわれている

図 3．2.1-3～図 3．2.1-4では、Cookie オブジェクトを CGI.header() 関数経由で出力する場合である。
図 3．2.1-4を見て分るとおり、クッキーの値は、URL エンコードされていることが確認できる。このようなコードを Perl プログラマが書く場合、「改行」や「;(セミコロン)」を特に意識せずに記述しても安全であるということである。

```
#! /usr/local/bin/perl
use strict;
use CGI;
use CGI::Cookie;
my ${cgi} = new CGI;
my ${str} = "test;¥nnewHeader:value";
my ${myC} = new CGI::Cookie(
        -name=>'cookieName',
        -value=>${str}          );
print ${cgi}->header(   -cookie=>${myC}       );
printf("${str}¥n");
printf(${cgi}->version());
printf("¥n");
__END__
```

図3．2.1-3：Cookie を CGI.header()関数経由で出力するテストスクリプト

```
# ./cgimod-test.pl
Set-Cookie: cookieName=test%3B%0AnewHeader%3Avalue; path=/
Date: Fri, 30 Mar 2007 04:53:21 GMT
Content-Type: text/html; charset=ISO-8859-1

test;
newHeader:value
2.752

# echo "HEAD /cgi-bin/cgimod-test.pl HTTP/1.0¥n¥n" | nc -nvv 127.0.0.1 80
(UNKNOWN) [127.0.0.1] 80 (?) open
HTTP/1.1 200 OK
Date: Fri, 30 Mar 2007 04:53:49 GMT
Server: Apache/2.0.47 (Unix) mod_ssl/2.0.47 OpenSSL/0.9.6f DAV/2
Set-Cookie: cookieName=test%3B%0AnewHeader%3Avalue; path=/
Connection: close
Content-Type: text/html; charset=ISO-8859-1

sent 40, rcvd 246
```

図3．2.1-4：図 3．2.1-3の実行結果。URL エンコードされたためヘッダインジェクションは発現しない

図 3．2.1-5～図 3．2.1-6では、CGI.redirect() 関数を使ったテストスクリプトである。
図 3．2.1-6を見て分るとおり、ローカルからの実行時には「”(ダブルクォート)」で囲まれつつも Lf インジェクションに成功しているが、Apache 経由では Location ヘッダ自体が HTTP レスポンスに含まれていないことが分る。
よって、CGI.redirect() 関数を使う限りにおいて、Lf インジェクションは気にする必要がないと考えられる。

```
#! /usr/local/bin/perl
use strict;
use CGI;
my ${cgi} = new CGI;
my ${str} = "test;¥nnewHeader:value";
print ${cgi}->redirect(${str});
printf("${str}¥n");
printf(${cgi}->version());
printf("¥n");
__END__
```

図3．2.1-5：CGI.redirect()**関数を使うテストスクリプト**

```
# ./cgimod-test.pl
Status: 302 Moved
location="test;
newHeader:value"

test;
newHeader:value
2.752
# echo "HEAD /cgi-bin/cgimod-test.pl HTTP/1.0¥n¥n" | nc -nvv 127.0.0.1 80
(UNKNOWN) [127.0.0.1] 80 (?) open
HTTP/1.1 302 Moved
Date: Fri, 30 Mar 2007 05:07:09 GMT
Server: Apache/2.0.47 (Unix) mod_ssl/2.0.47 OpenSSL/0.9.6f DAV/2
Connection: close
Content-Type: text/html; charset=iso-8859-1

sent 40, rcvd 189
```

図3．2.1-6：図 3．2.1-5**の実行結果。**

ローカル実行した時は「"(ダブルクォート)」で囲まれながらも Lf **インジェクションに成功しているが、**

Apache **経由では、**HTTP **レスポンスに含まれていない**

ローカルで「"」で囲まれているところ、「名」と「値」の境界が「:」でなくて「=」になっているところが興味深い。

図 3．2.1-7～図 3．2.1-8では、CGI.header() 関数を使ったテストスクリプトである。
図 3．2.1-8を見て分るとおり、ローカルからの実行時には「”(ダブルクォート)」で囲まれつつも Lf インジェクションに成功しているが、Apache 経由では「500」エラーとなっていることが分る。
よって、CGI.header() 関数を使う限りにおいて、Lf インジェクションは気にする必要がないと考えられる。

```
#! /usr/local/bin/perl
use strict;
use CGI;

my ${cgi} = new CGI;
my ${str} = "test;¥nnewHeader:value";

print ${cgi}->header(
                -myHeader=>${str}
);

printf("${str}¥n");
printf(${cgi}->version());
printf("¥n");
__END__
```

図3．2.1-7：CGI.header()関数を使うテストスクリプト

```
# ./cgimod-test.pl
myheader="test;
newHeader:value"
Content-Type: text/html; charset=ISO-8859-1

test;
newHeader:value
2.752

# echo "HEAD /cgi-bin/cgimod-test.pl HTTP/1.0¥n¥n" | nc -nvv 127.0.0.1 80
(UNKNOWN) [127.0.0.1] 80 (?) open
```

```
HTTP/1.1 500 Internal Server Error
Date: Fri, 30 Mar 2007 05:19:21 GMT
Server: Apache/2.0.47 (Unix) mod_ssl/2.0.47 OpenSSL/0.9.6f DAV/2
Connection: close
Content-Type: text/html; charset=iso-8859-1

sent 40, rcvd 205
```

図3．2.1-8：図 3．2.1-7の実行結果。

ローカル実行した時は「"(ダブルクォート)」で囲まれながらも Lf インジェクションに成功しているが、

Apache 経由では、HTTP レスポンスに含まれていない

ローカルで「"」で囲まれているところ、「名」と「値」の境界が「:」でなくて「=」になっているところが興味深い。

また、Apache 経由の時「500」エラーが返されている

結論：

CGI.pm モジュールの header() 関数、redirect()関数を介して HTTP レスポンスを作る限りにおいて、Perl プログラマは CrLf インジェクションについて気にかける必要はない。
また、クッキー情報を CGI::Cookie オブジェクトで操作するかぎりにおいて、本項で指摘してる「改行」「;(セミコロン)」について気にかける必要はない。

3．2.2　Perl5.005_03/CGI.pm2.46 の場合

この章では､筆者の ISP 環境(Perl5.005_03/CGI.pm2.46)を実験環境として使用した結果である。

3．2.1と同一の結果であった。

```
C:¥>type a.txt
HEAD /~example/cgi-bin/cgimod-test.pl HTTP/1.0
Host: www.example.com

                    // ここは CGI.header() の結果である

C:¥>nc www.example.com 80 < a.txt
HTTP/1.1 500 Internal Server Error
```

Date: Fri, 30 Mar 2007 05:59:58 GMT

Server: Apache/1.3.9 (Unix)

Connection: close

Content-Type: text/html

// ここは CGI.redirect() の結果である

C:¥>nc www.example.com 80 < a.txt

HTTP/1.1 ***302*** Moved

Date: Fri, 30 Mar 2007 06:01:49 GMT

Server: Apache/1.3.9 (Unix)

Connection: close

Content-Type: text/html

// ここは CGI.header() 経由の CGI::Cookie オブジェクトの結果である

C:¥>nc www.example.com 80 < a.txt

HTTP/1.1 200 OK

Date: Fri, 30 Mar 2007 06:05:43 GMT

Server: Apache/1.3.9 (Unix)

Set-Cookie: cookieName=***test%3B%0AnewHeader%3Avalue***; path=/~example/cgi-bin/

Connection: close

Content-Type: text/html

// ここは CGI.header() 経由しないの CGI::Cookie オブジェクトの結果である

C:¥>nc www.example.com 80 < a.txt

HTTP/1.1 200 OK

Date: Fri, 30 Mar 2007 06:07:10 GMT

Server: Apache/1.3.9 (Unix)

Set-Cookie: cookieName=***test%3B%0AnewHeader%3Avalue***; path=/~example/cgi-bin/

Connection: close

Content-Type: text/html

図3．2.2-1：ISP の環境なので、ローカルから実行した結果は不明だが、3．2.1章と同一の結果となった

3.3 IIS + ASP の場合

IIS + ASP での HTTP ヘッダ操作は、以下の関数がよく使われる。

関数名	操作内容
Response.Redirect	Web ページのリダイレクト要求
Response.Cookies	クッキー情報の送信
Response.AddHeader	任意ヘッダの追加

これらの関数に汚染データが配置されるようなテストプログラム(図 3.3-1)を作成し、Netcat を使ってどのような HTTP レスポンスが返るかを観察した。

MS-WindowsServer2003 SP1(日本語版)を実験環境として使用した。

```
<%
 Option Explicit
 Dim str
 str = Request.QueryString("inputData")
 Response.AddHeader "TestHeader",str
 Response.Cookies("TestCookie") = str
 Response.Redirect str
%>
```

図3.3-1 : ASP **のヘッダ送信関数の実験プログラム**

図3.3-2：図 3.3-1**の結果**

結果は、図 3.3-2**のように、**AddHeader() **関数以外は関数内部で改行コードのサニタイズ処理が行われていることが確認された。**
よって、ASP **で上記の関数を使う場合、**AddHeader() **関数以外は** HTTP **ヘッダ・インジェクションに気を使わずにコーディングを行うことができる。**
また、AddHeader() **関数を使用する際には、**ASP **プログラム内で改行コードをサニタイズ処理(他の関数との関係も考えれば、**URL **エンコードが妥当だろう)することがセキュリティ対策上必要な対策である、ということである。**

蛇足:

- **図** 3.3-2**のように、**ASP **では、改行コード(**Cr **と** Lf)**などが、**URL **エンコードされている。**
- Location **ヘッダには、汚染データそのものが与えられている。汚染データに別サイトの** URL **を指定される必要がない場合は、その対策が別途必要である。**
- **クッキー情報の箇所では「**;**」がサニタイズ処理されている(「**;**」が** URL **エンコードされ「**%3B**」になっている)。よって、**ASP.NET **とは異なり、クッキー情報の値をセットする際には、とくにセキュリティ対策を考慮しなくてもよい。ということになる。**

3.4 ASP.NET の場合

ASP.NET での HTTP ヘッダ操作は、以下の関数がよく使われるだろう。

関数名	操作内容
Response.RedirectLocation	Web ページのリダイレクト要求
Response.Cookies.Add	クッキー情報の送信
Response.AddHeader Response.AppendHeader	任意ヘッダの追加

これらの関数に汚染データが渡るようなテストプログラム(図 3.4-1)を作成し、Netcat を使ってどのような HTTP レスポンスが返るかを観察した。

MS-Windows2000 SP4(日本語版)上の WebMatrix0.6.812 + .Net Framework1.1SP1(1.1.4322)を実験環境として使用した。

```
// ヘッダ追加用メソッド
Response.AddHeader("myAddHeader",TextBox1.Text);
Response.AppendHeader("myAppendHeader",TextBox1.Text);
// クッキー送出用メソッド
HttpCookie MyCookie = new HttpCookie("myAddCookie");
MyCookie.Value = TextBox1.Text;
Response.Cookies.Add(MyCookie);
// リダイレクト用メソッド
Response.RedirectLocation = TextBox1.Text;
```

図3.4-1 : ASP.NET **のヘッダ送信関数の実験プログラム** (C#)

図3.4-2： 図 3.4-1**の結果**

結果は、図 3.4-2**のように、関数内部で改行コードのサニタイズ処理が行われていることが確認された。**
よって、ASP.NET **で上記の関数を使う場合は、**HTTP **ヘッダ・インジェクションに気を使わずにコーディングを行うことができる。**

蛇足:

- **図** 3.4-2**のように、**ASP.NET **では、改行コード(**Cr **と** Lf**)が、**URL **エンコードされている。**
- Location **ヘッダには、汚染データそのものが与えられている。汚染データに別サイトの** URL **が指定される必要がない場合は、その対策が別途必要である。**
- **クッキー情報の箇所では「;」がサニタイズ処理されていない。よって、クッキー情報の値以外のプロパティ(**Domain **や** Expires **など)を汚染データによってセットされる危険性がある。他の関数との関係も考えれば、「;」→「%3b」に置換する** URL **エンコード処理を** ASP.NET **プログラマは行う必要がある。**

3.5 JavaServlet の場合

JavaServlet での HTTP ヘッダ操作は、以下の関数がよく使われるだろう。

関数名	操作内容
response.sendRedirect	Web ページのリダイレクト要求
response.addCookie	クッキー情報の送信
response.addHeader	任意ヘッダの追加

これらの関数に汚染データが渡るようなテストプログラム(図 3.5-1)を作成し、Netcat を使ってどのような HTTP レスポンスが返るかを観察した。

MS-WindowsXP SP2(日本語版)上の Tomcat5.5.17、JDK1.5.0_06、Eclipse3.1.2 を実験環境として使用した。

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
            String myAddStr;
            String IsRedirect;
            request.setCharacterEncoding("Windows-31J");
            myAddStr = request.getParameter("addStr");
            IsRedirect = request.getParameter("IsRedirect");
            // HTTP ヘッダの追加
            response.addHeader("myAddHeader",myAddStr);
            // クッキーの追加
            Cookie myCookie = new Cookie("myCookie",myAddStr);
            response.addCookie(myCookie);
            // location ヘッダの追加
            // redirect to JSP
            if(IsRedirect.equals("1") == true){
                        getServletConfig().getServletContext().getRequestDispatcher("/result.jsp").forwar
d(request,response);
            }else{                response.sendRedirect(myAddStr);          }          }
```

図3.5-1 : JavaServlet **のヘッダ送信関数の実験プログラム**

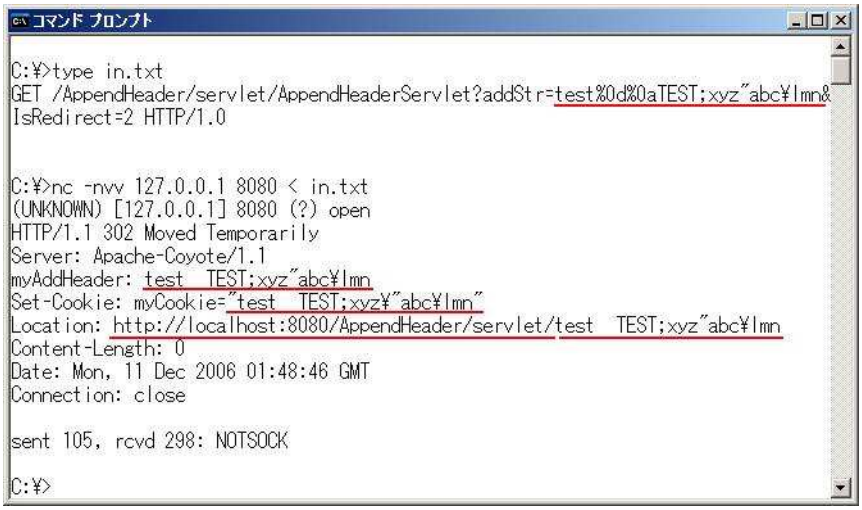

図3.5-2：図 3.5-1**の結果**

(パケットキャプチャを行うと、改行コードが「0x20**」に置換されていることが分かる)**

結果は、図 3.5-2**のように、関数内部で改行コードのサニタイズ処理が行われていることが確認された。**
よって、JavaServlet **で上記の関数を使う場合は、**HTTP **ヘッダ・インジェクションに気を使わずにコーディングを行うことができる。**

蛇足:

- **改行コード(**Cr **と** Lf**)がどのコードに置換されるかというものは、パケットキャプチャをした結果、「半角スペース(**0x20**)」に置換されることを確認した。**
- Location **ヘッダの左側の赤線にあるように、自動で自サイトの** URL **を先頭に挿入するようである。この現象は汚染データを使った** URL **リダイレクトを自** Web **アプリケーション内に限定できる、ということである。**
- **クッキー情報の** HTTP **ヘッダでは、入力データを「”」で囲むことが、図** 3.5-2**で確認できる。また、「”」は「¥”」でエスケープするようであるが、エスケープ文字「¥」はエスケープしなくてもいいようである。**

3.6 PHP5.2.0 の場合

PHP での HTTP ヘッダ操作は、以下の関数がよく使われるだろう。

関数名	操作内容
header(“Location: <<入力データ>>”)	Web ページのリダイレクト要求
setcookie setrawcookie	クッキー情報の送信
header	任意ヘッダの追加

リダイレクトには、header() 関数を使って、Location ヘッダを作成するという方法を用いる。setrawcookie()関数は、URL エンコードしないという点で setcookie() 関数と同一であるとマニュアルには記載されている。

これらの関数に汚染データが渡るようなテストプログラム(図 3.6-1)を作成し、Netcat を使ってどのような HTTP レスポンスが返るかを観察した。

MS-WindowsServer2003 SP1(日本語版)上の PHP5.2.0 を実験環境として使用した。

```
<?
        $str = $_REQUEST["inputData"];
        setcookie('cookiename', $str, time() + 60);
        setrawcookie('RAWcookiename', $str, time() + 60);
        header("myHeader: $str");
?>
```

図3.6-1 : PHP **のヘッダ送信関数の実験プログラム**

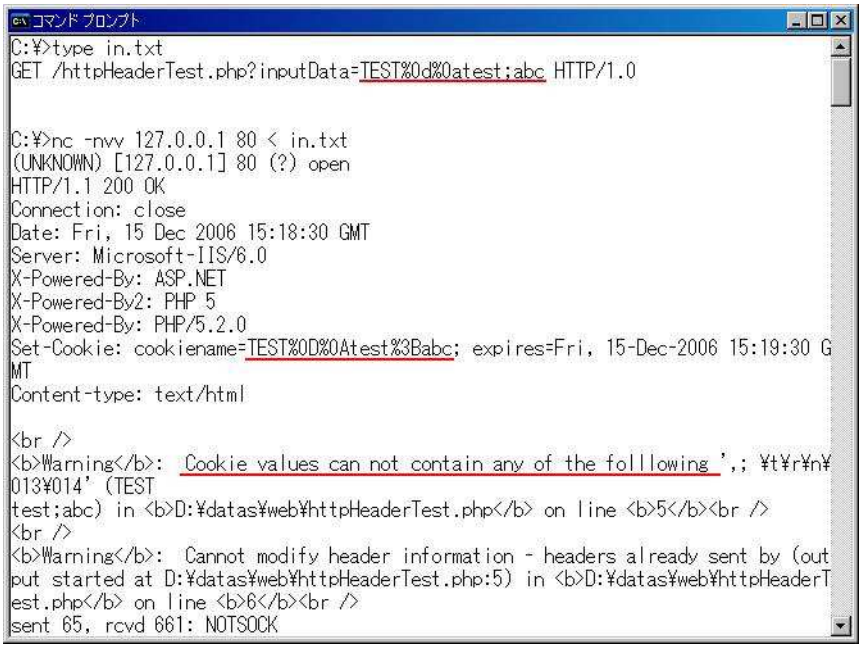

図3.6-2： 図 3.6-1**の結果**

クッキーへの出力は URL **エンコードされている**(setcookie)。setrawcookie() **関数ではエラーになる**

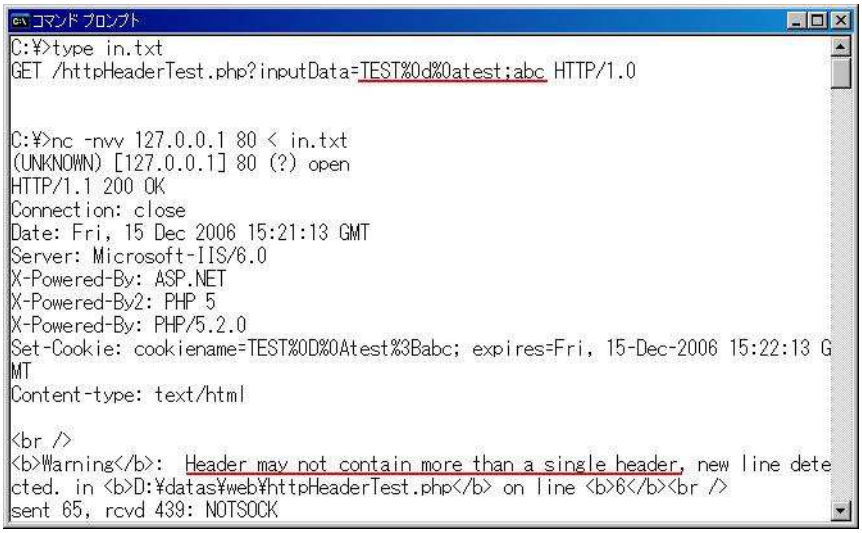

図3.6-3： 図 3.6-1**の結果**

header() **関数に改行コードを与えるとエラーになる**

結果は、図 3.6-2**～図** 3.6-3**のように、**setcookie() **関数では** URL **エンコードされ、またそれ以外の** setrawcookie() **関数、**header() **関数では、エラーとなることが確認された。**
よって、PHP **で上記の関数を使う場合は、**HTTP **ヘッダ・インジェクションに気を使わずにコーディングを行うことができる。**

蛇足:

- setrawcookie() **関数、**header() **関数に改行コードを与えることでエラーとなる。**
  エラーにならないようにするには、PHP **プログラム中で、改行コードを事前に** URL **エンコードするなど** PHP **プログラマが明示的に処理することで対処する必要がある。**
- setcookie() **関数は改行コードだけではなく、クッキー情報のプロパティのデリミタである「**;(**セミコロン**)**」も** URL **エンコードされることが確認できる。**

3.7 AzaraC0.3.0 の場合

AzaraC は、C++用の CGI テンプレートである。これについても調査してみた。

AzaraC では、クラス azarac::HTTPResponse のメソッドとして以下が提供されている

メソッド名	操作内容
addHeader(キー名,値)	任意ヘッダの追加
setHeader (キー名,値)	任意ヘッダの上書き
addCookie(クッキークラス)	クッキー情報の送信

クッキークラスとは、azarac::Cookie クラスのインスタンスを指定する。

これらの関数がどのような HTTP ヘッダを送信するかを以下で確認してみた。
利用したソフトウェアは、RedHat7.3、g++ 2.96、AzaraC 0.3.0 である。

```
#include <iostream>
#include "azarac.h"
using namespace azarac;
int main(){
  if (display("./headerTest.tpl")<0) cout<<what();    }
```

図3.7-1 : headerTest.cpp (**ロジック部の** C++ **ソース**)

```
<%@ page Content-Type="text/html;" %>
<% response.addHeader("abc","xyz"); %>
<% response.setHeader("abc","lmn\r\nstu"); %>
<% response.addHeader("xyz","abc\r\nlmn"); %>
<% response.addHeader("test","end"); %>
<%
 Cookie new_cookie("myCookie","atai;CRLF\r\nva=lue");
 response.addCookie(new_cookie);
 response.addHeader("zzz","last");                    %>
<html><body>Hello, World!</body></html>
```

図3.7-2 : headerTest.tpl (**デザイン部のテンプレート**)

```
Set-Cookie: myCookie=atai%3BCRLF%0D%0Ava%3Dlue;
Content-Type: text/html;
abc: lmn
stu
test: end
xyz: abc
lmn
zzz: last


<html><body>Hello, World!</body></html>
```

図3.7-3：図 3.7-1**と図** 3.7-2**を** g++**で**-lazarac **オプションでコンパイルした実行ファイルを実行結果**
Cookie **に関しては** URL **エンコードされているが、**
setHeader()/addHeader()**メソッドで指定した部分は改行コードがそのままである**

結果は、図 3.7-3**のように、**addHeader() / setHeader()**メソッド関数共に関数内部で改行コードのサニタイズ処理が行われていないことが確認された。**
しかしながら Cookie **に関しては、**URL **エンコードが行なわれていることが確認できる。**
よって、AzaraC **の** addHeader()/setHeader()**メソッドを使う場合、アプリケーション・プログラマが改行コードのサニタイズ処理をしたうえで呼び出す必要がある。**

4　執筆者など

4.1 本文書の免責事項

本文書に記述されてる情報の利用は、読者の責任に帰するものとする。
本文書で行った実験は、執筆者の実験環境で確認したものである。モジュールの細かいバージョンなどの状況によっては、現象が異なる可能性がある。

4.2 執筆者

■ active@window.goukaku.com

4.3 更新履歴

最初のバージョン : 2007 年 03 月 05 日
ver1.1 : 2007 年 03 月 30 日
3.2, 3．2.1, 3．2.2 を追加した
ver1.2 : 2007 年 10 月 30 日
3.7 を追加した

4.4 本文書の最新バージョン

http://rocketeer.dip.jp/secProg/index.htm

4.5 HTTP ヘッダ名

本文書では除外しているが、HTTP ヘッダ名に汚染データを割り当てる場合は、ヘッダ名とヘッダ値のデリミタが「:(コロン)」である。そのような場面では「:(コロン)」のサニタイズ処理が必要である。
また、クッキー情報では、クッキー名とクッキーの値が「=(イコール)」で区切られている。クッキー名に汚染データが入るような場合は、「=(イコール)」がどのように処理されるかシステム開発前に事前に観察しておく必要がある。

以　上