HTTP Status Code Pollution

ＮＴＴ コミュニケーションズ株式会社
ソリューションサービス部
第四エンジニアリング部門
セキュリティオペレーション担当

2012 年 06 月 15 日

Ver. 1.0

1. 調査概要

1.1. 調査概要

PHP や CGI では、HTTP Response のヘッダ情報を出力する関数を用いて、HTTP Response の Status Code を操作することができる。
このヘッダ情報を出力する関数に関しては、改行コード(Cr や Lf)をエスケープしていないために発現する「HTTP ヘッダ・インジェクション」または「CrLf インジェクション」が有名であるが、HTTP Response Status Code が汚染される場合について、本文書では検討した。

2. HTTP Response の Status Code汚染

2.1. PHP

他の Web アプリケーション・サーバでは、大抵の場合は、HTTP Response の Status Code を操作するための専用の関数が用意されていると思うが、PHP では、header()関数を用いて HTTP Response の Status Code を操作する。
つまり、Webアプリケーションとして、HTTP Response Status Codeとして「401 Unauthorized」を出力したい場合、図 2.1-1のように、HTTP Response のヘッダ情報を出力する関数を用いて行う。

```
<?php
                    ・・・
header("HTTP/1.1 401 Unauthorized");
                    ・・・
?>
```

図 2.1-1 : PHP では、header 関数を用いて Status Code を操作する

このように、PHP では、Status Code を操作する関数と、ヘッダ情報を操作する関数が分離していない。
つまり、ヘッダ情報を操作する関数が分離していないという事は、不正行為者がこの header() 関数に与える引数の先頭から汚染可能である場合、任意のヘッダ情報がインジェクションされるだけではなく、HTTP Response の Status Code を上書きすることができることを意味している。

2.2. CGI

実は、CGIでも、HTTP Response のStatus Codeを操作することができる(図 2.2-1)。
CGI プログラムが省略した場合、HTTP Response Status Code は「200 OK」となる。

```
#! /usr/bin/php
<?php
                    ・・・
echo "Status: 401¥n";
echo "Content-Type: text/html¥n¥n";
                    ・・・
?>
```

図 2.2-1 : CGI(言語は PHP だが)CGI の場合、
最初の空行までに「Status:」で始まる行を出力することによって、
Status Code を操作することができる

2.3. 脆弱なWebページを想定する(mod_php使用のPHP)

以下のような Web ページを想定してみる

- クエリ文字列「str1」を受け取り、Web ブラウザ上に表示する
- クエリ文字列「str2」を受け取り、HTTP Response ヘッダとして出力する
- クエリ文字列「str1」に「<」が含まれていた場合、Status Code を操作して、「login.html」へリダイレクトさせる
- クエリ文字列「str1」に「<」が含まれていた場合、リダイレクトさせるので、そのまま HTTP Response のボディに格納する。
  「Status Code=3xx」なので、HTTP Response のボディは Web ブラウザ上に表示されないため、仮に XSS(クロスサイト・スクリプティング)脆弱性があっても、発現しない

PHP のソースコードは、図 2.3-1である。

```
<?php
if(!empty($_GET["str1"]) && !empty($_GET["str2"])){
 if(preg_match("/</",$_GET["str1"])){
  header("Location: ./login.html");
 }
 header($_GET["str2"]);
 echo $_GET["str1"];
}
?>
```

図 2.3-1 : PHP(mod_php)で書かれた実証テスト用スクリプト

2.4. 脆弱なWebページを想定する(CGI(言語はPHP))

「2.3 脆弱なWebページを想定する(mod_php使用のPHP)」と同じ使用のCGIも作成した(図 2.4-1)。

```
#!/usr/bin/php
<?php
echo "Content-Type: text/html¥n";
$var = $_SERVER["QUERY_STRING"];
$splite_var_arr = explode("&",$var);
for($i=0;$i<count($splite_var_arr);$i++){
 $value_arr = explode("=",$splite_var_arr[$i]);
 $key = $value_arr[0];
 $val = $value_arr[1];
 $query_string_value_arr[$key] = $val;
}
echo urldecode($query_string_value_arr["str2"]) . "¥n";
if(preg_match("/</",$query_string_value_arr["str1"])){
 echo "Location: ../test/login.html¥n";
}
echo "¥n";
$query_string_value_arr["str2"] =
str_replace(array("¥r","¥n"),"",$query_string_value_arr["str2"]);
echo urldecode($query_string_value_arr["str1"]);
?>
```

図 2.4-1 : CGI の場合の実証テスト用スクリプト

2.5. 実証テスト結果(mod_php使用のPHP)

実際に、図 2.3-1に対して試験を行ったキャプチャ図である

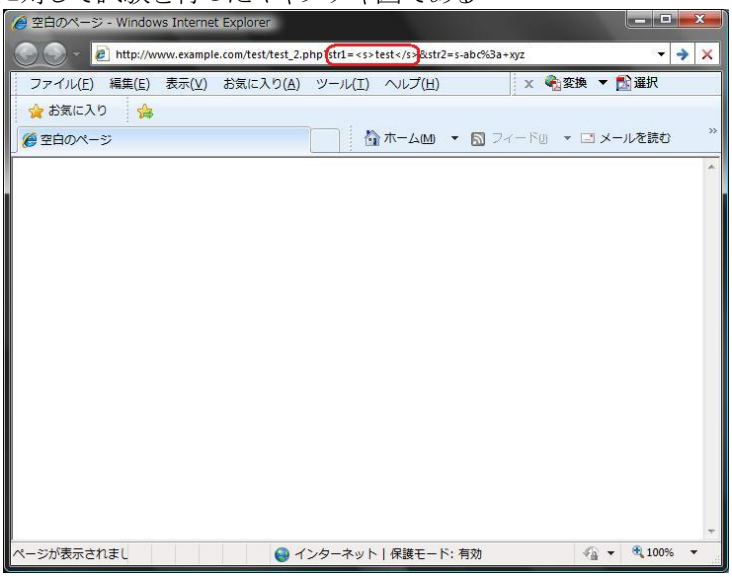

図 2.5-1 : クエリ文字列「str1」に XSS 試験用文字列を与える

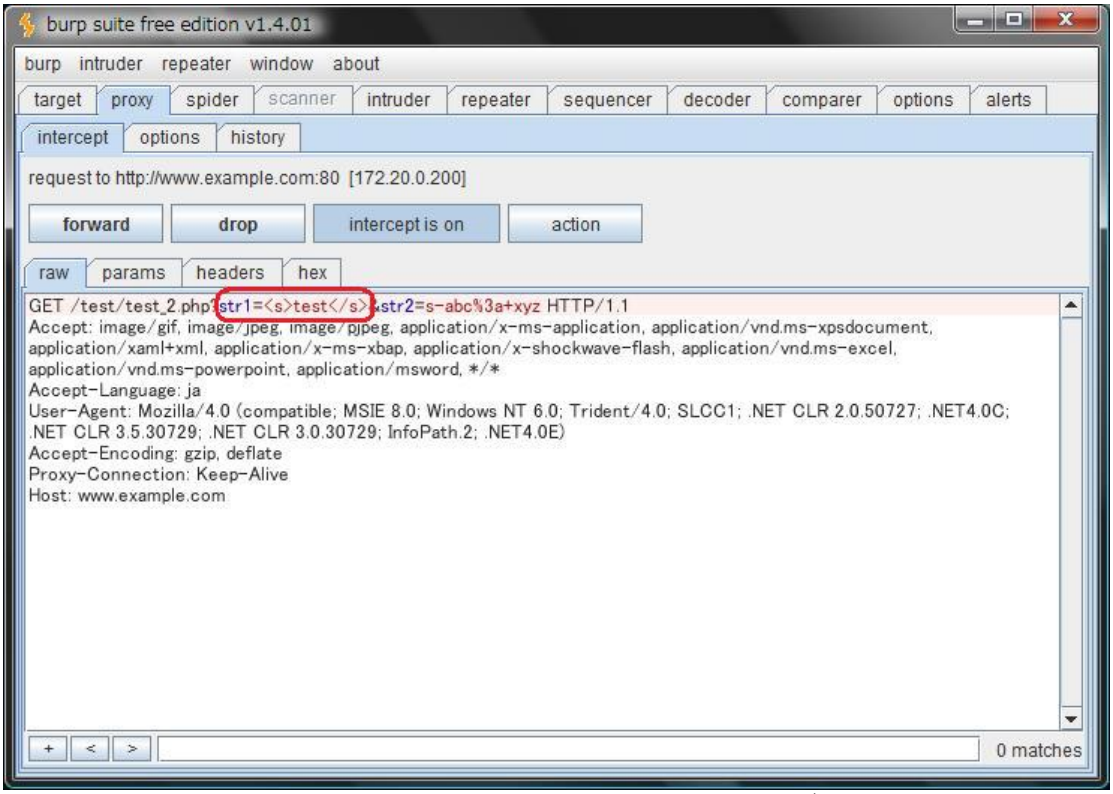

図 2.5-2 : 図 2.5-1のHTTPリクエスト・メッセージ

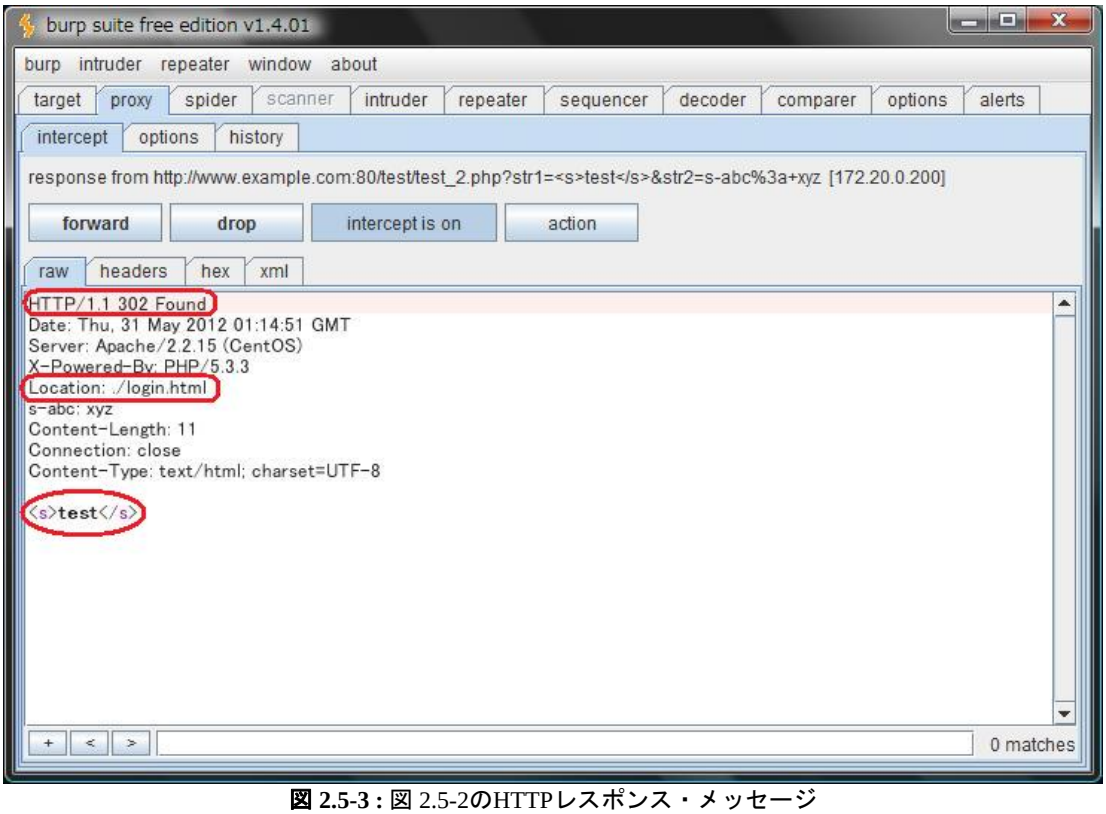

図 2.5-3 : 図 2.5-2のHTTPレスポンス・メッセージ

ボディ部分はXSSが発現する状態になっているが・・・

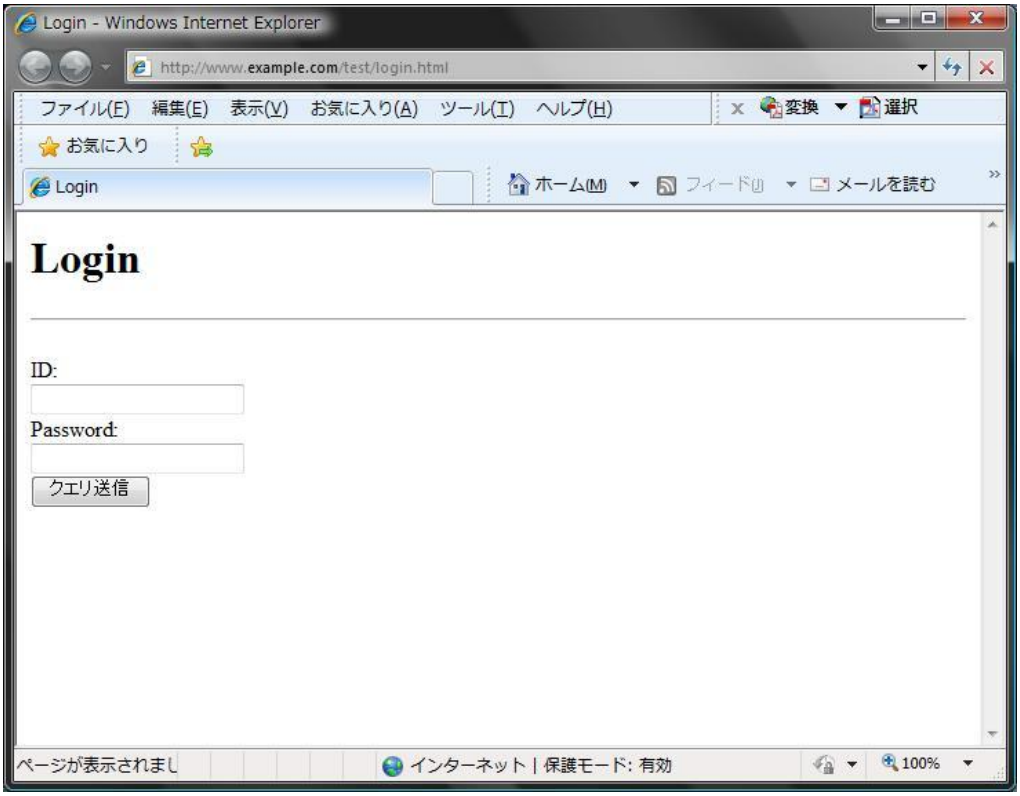

図 2.5-4 : 図 2.5-3を受領した結果。

Webブラウザは「login.html」へリダイレクトされるため、XSSは発現しない

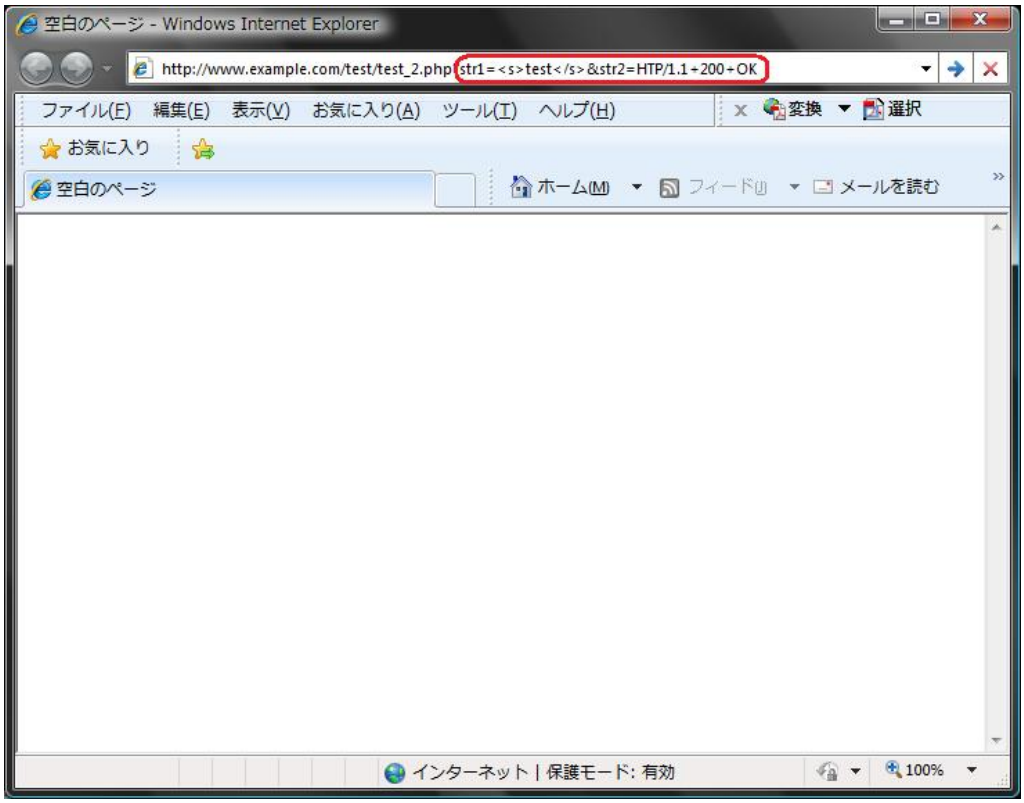

図 2.5-5 : 次に「str1」に XSS 試験用文字列を与えつつ、

変数「str2」には HTTP Response Status Code を改ざんするような値を与えてみる

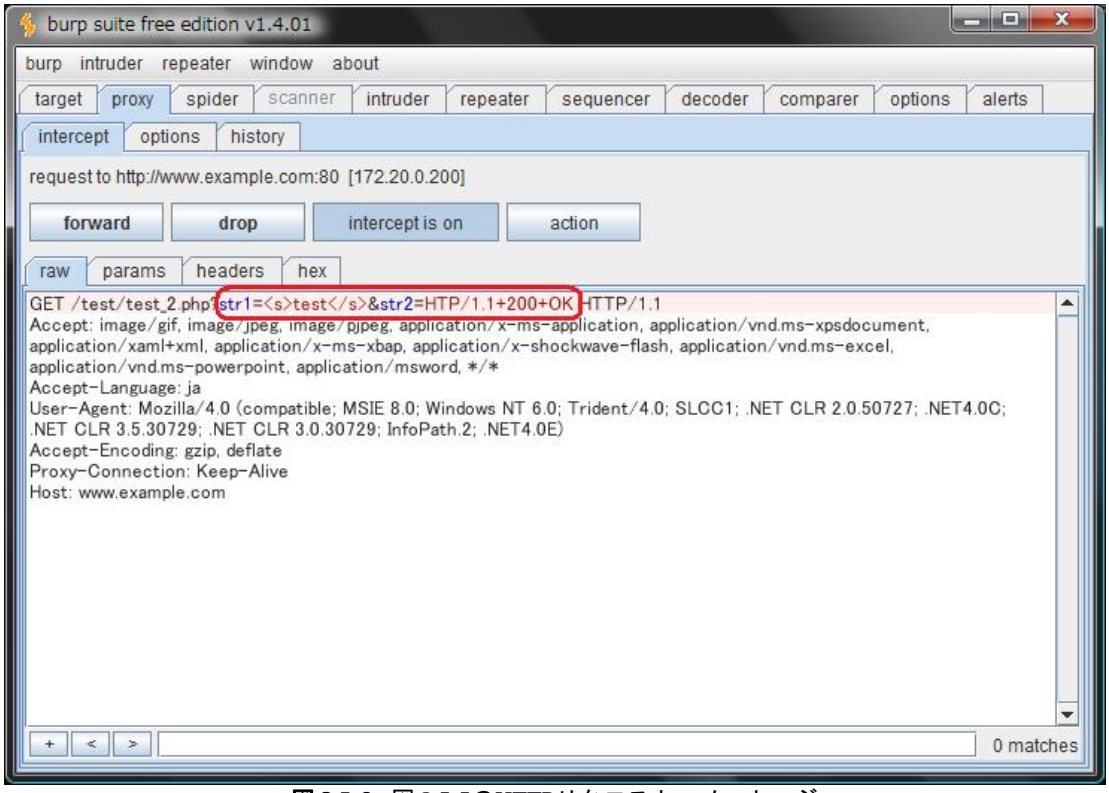

図 2.5-6 : 図 2.5-5のHTTPリクエスト・メッセージ

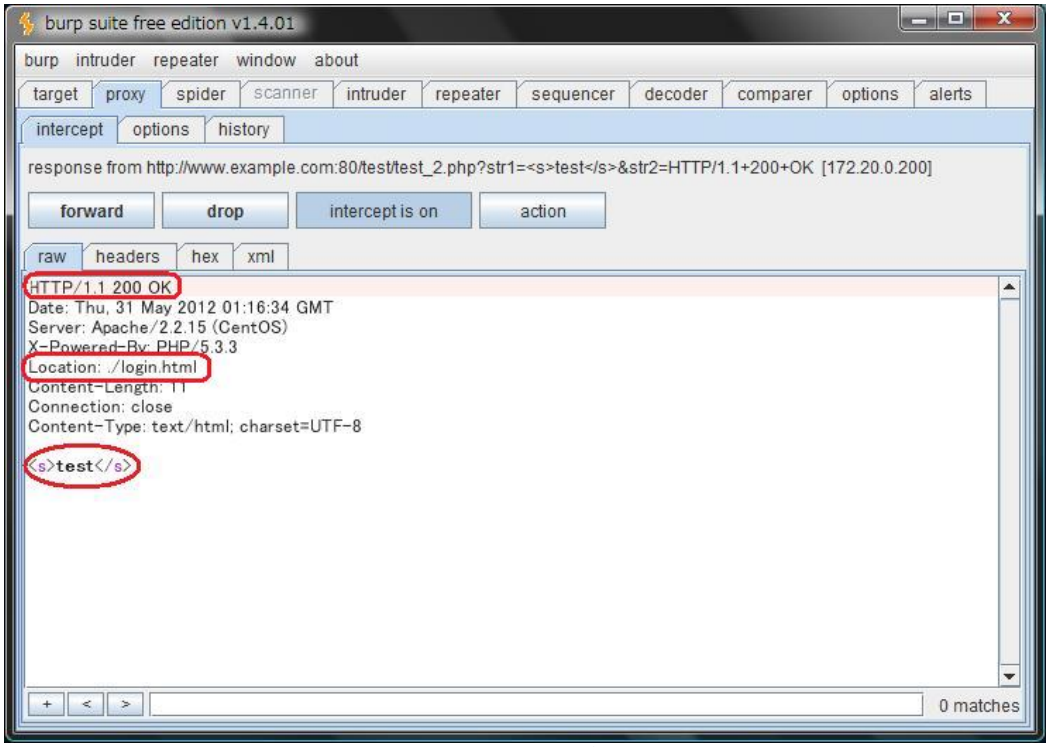

図 2.5-7 : 図 2.5-6のHTTPレスポンス・メッセージ

ボディ部分はXSSが発現する状態になっており、

さらにはHTTP Response Status Codeも改ざんされている

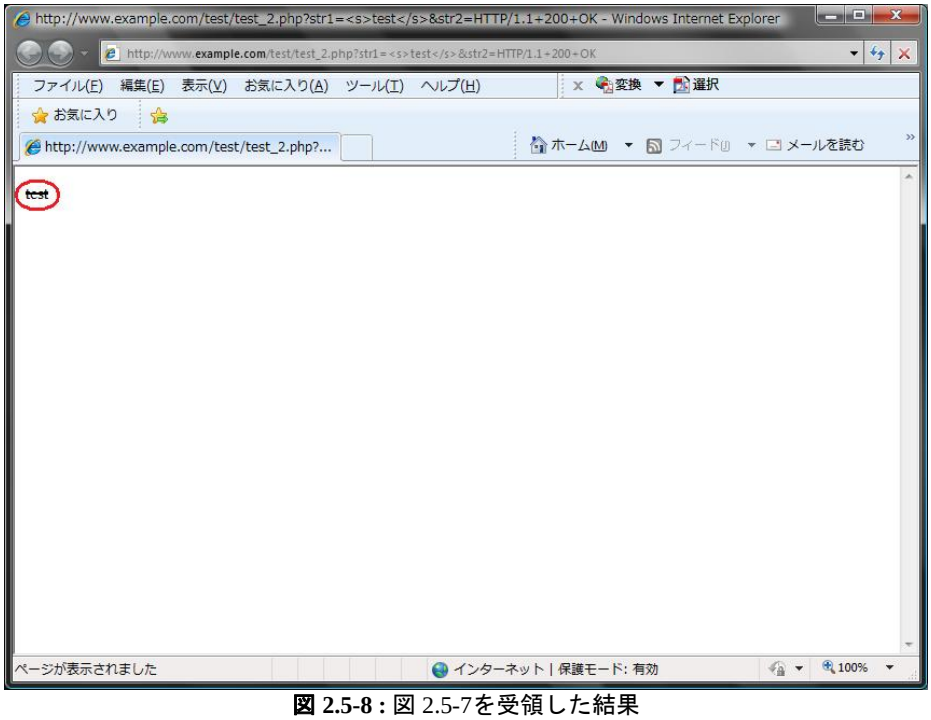

図 2.5-8 : 図 2.5-7を受領した結果

Webブラウザは「login.html」へリダイレクトされず、XSSが発現する

2.6. 実証テスト結果(CGI(言語はPHP))

次は、図 2.4-1に対して行った試験結果のキャプチャ図である。

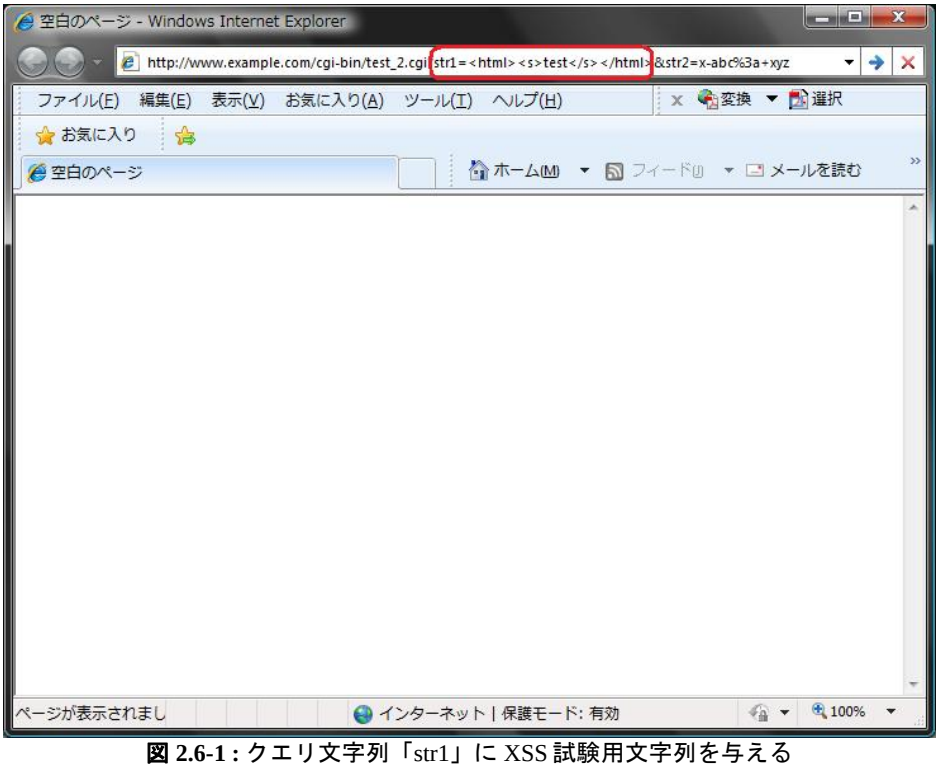

図 2.6-1 : クエリ文字列「str1」に XSS 試験用文字列を与える

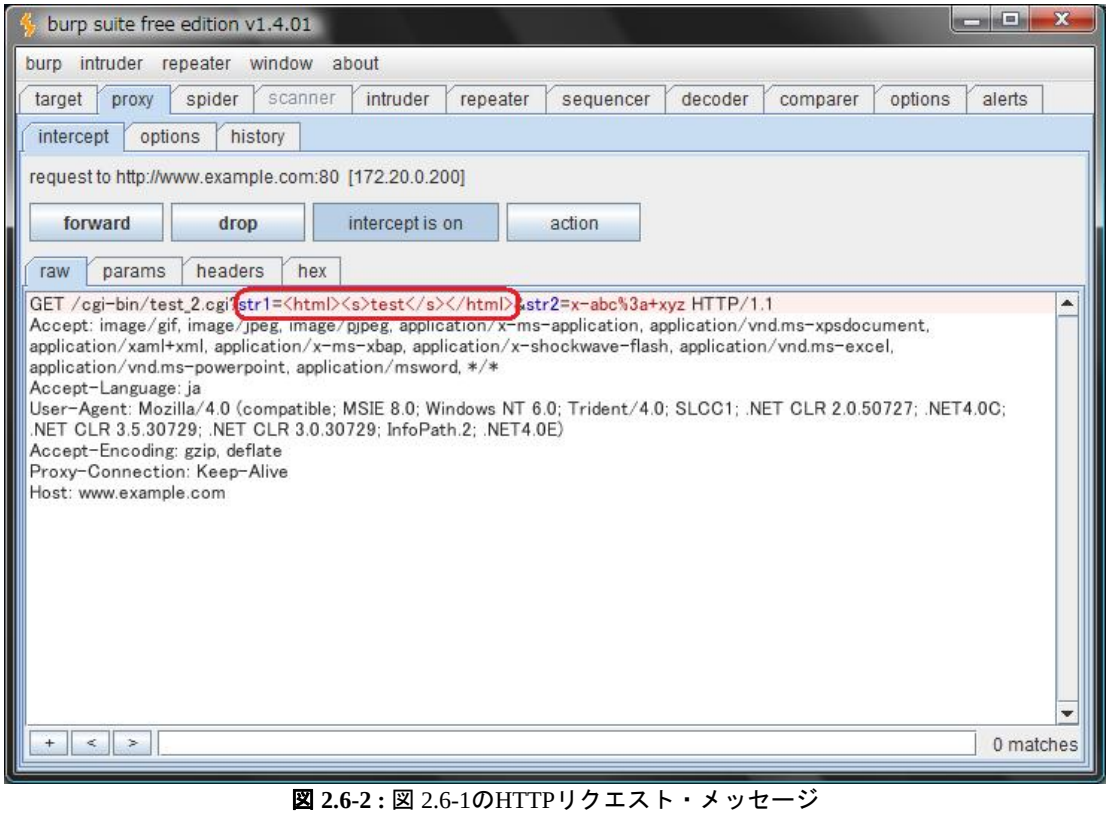

図 2.6-2 : 図 2.6-1のHTTPリクエスト・メッセージ

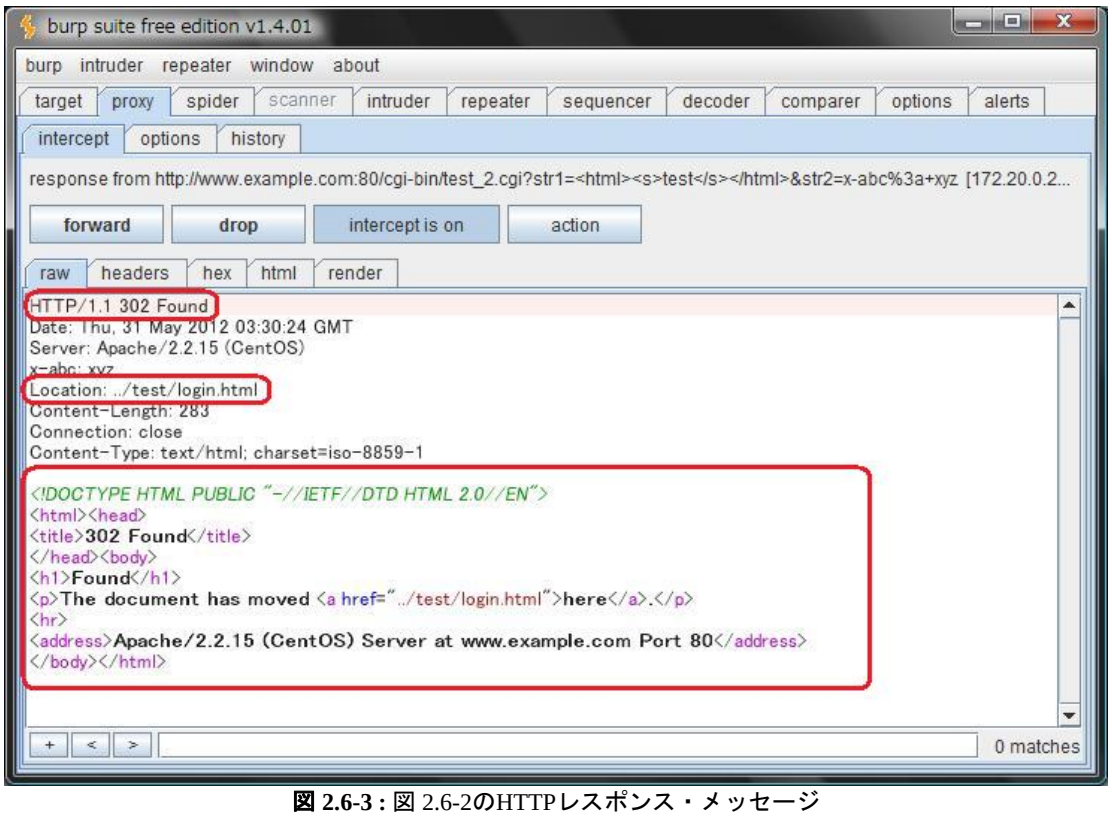

図 2.6-3 : 図 2.6-2のHTTPレスポンス・メッセージ

CGIの場合、図 2.5-3と異なり、HTTPレスポンス・ボディ部分が上書きされてしまっている

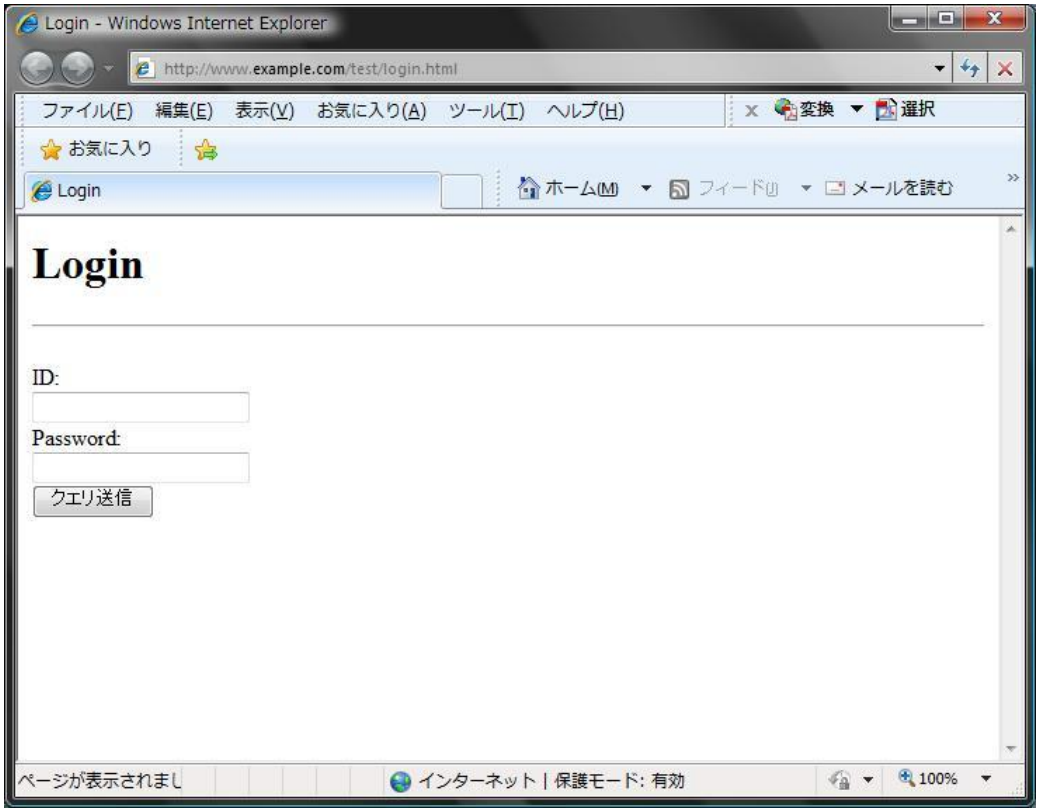

図 2.6-4 : 図 2.6-3を受領した結果。

Webブラウザは「login.html」へリダイレクトされるため、XSSは発現しない

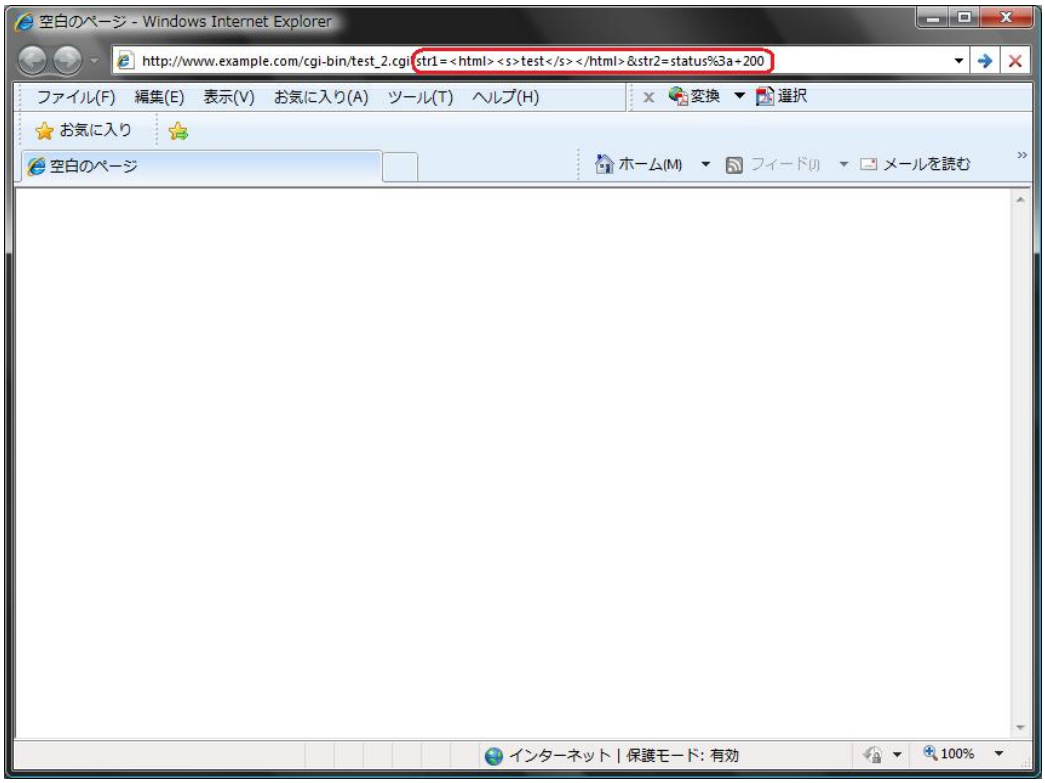

図 2.6-5 : 次に「str1」に XSS 試験用文字列を与えつつ、

変数「str2」には HTTP Response Status Code を改ざんするような値を与えてみる

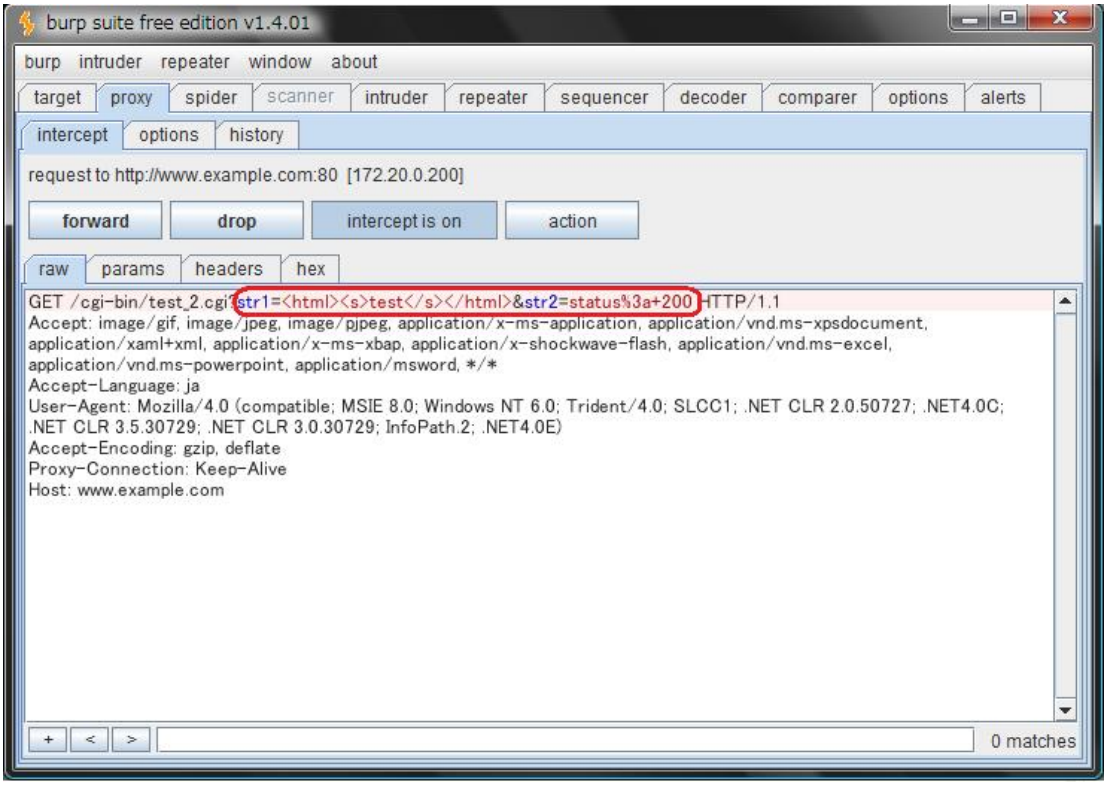

図 2.6-6 : 図 2.6-5のHTTPリクエスト・メッセージ

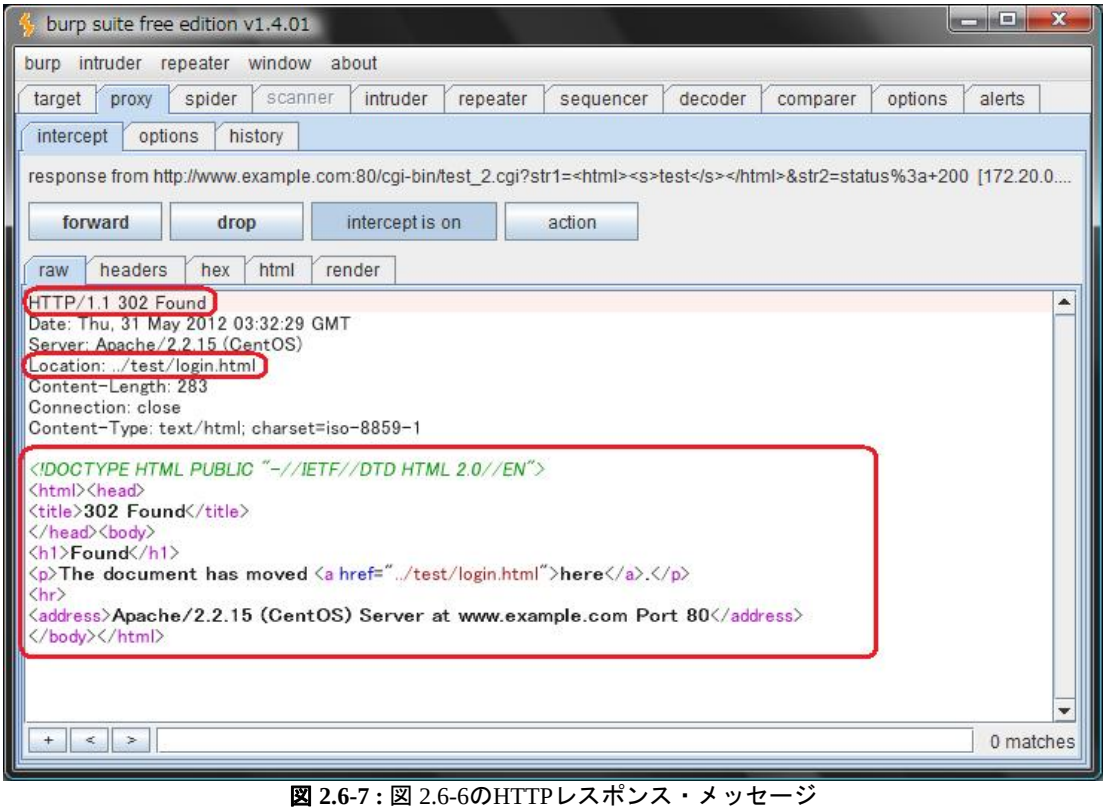

図 2.6-7 : 図 2.6-6のHTTPレスポンス・メッセージ

CGIの場合、「Location」ヘッダ付きでは「Status Code⇒200」への書き換えはできないようだ

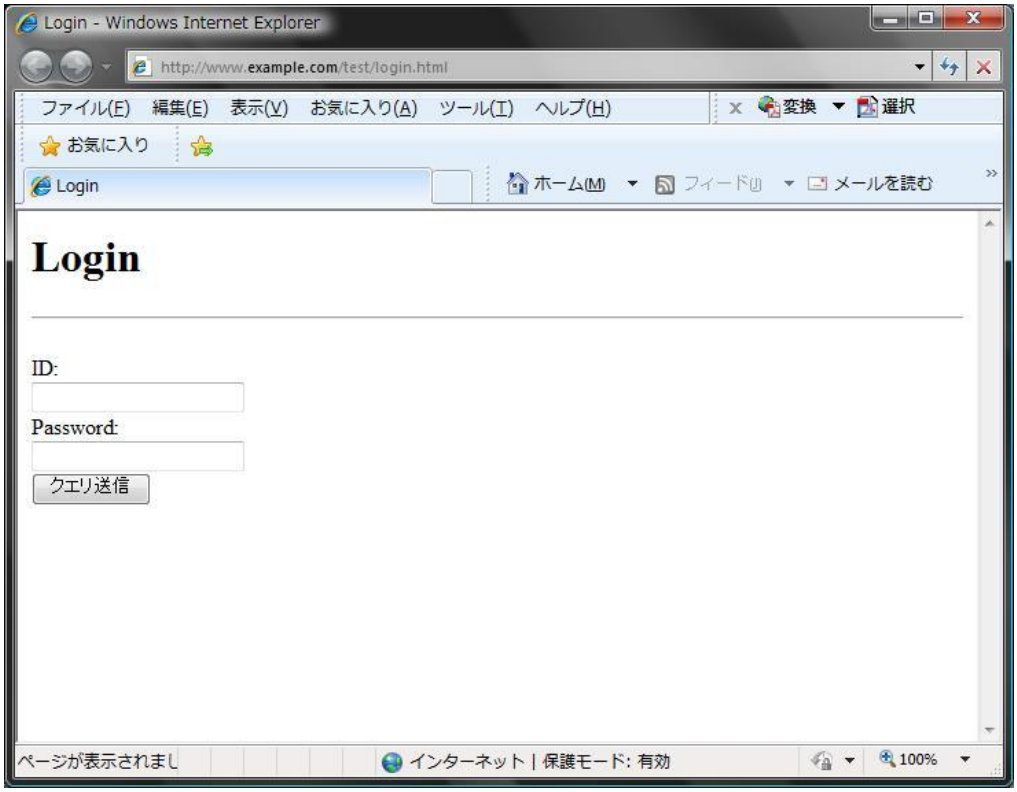

図 2.6-8 : 図 2.6-7を受領した結果

Webブラウザは「login.html」へリダイレクトされるため、XSSは発現しない

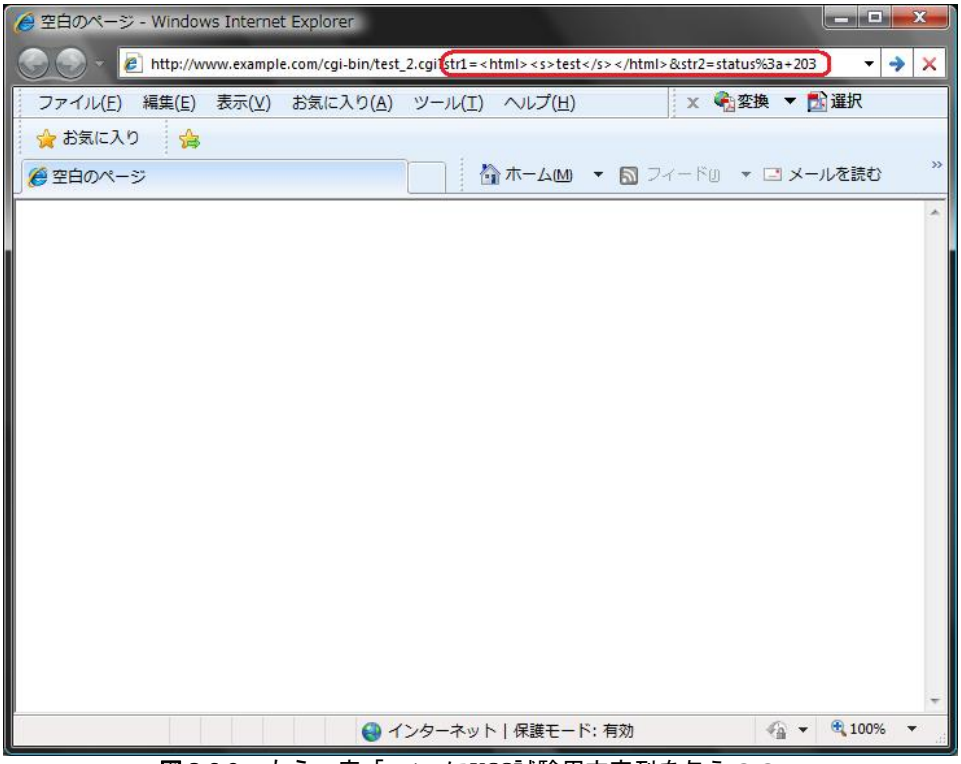

図 2.6-9 : もう一度「str1」にXSS試験用文字列を与えつつ、

変数「str2」にはHTTP Response Status Codeを改ざんするような値を与えてみる

図 2.6-5と異なり「Status Code=200」以外の値となるようにする

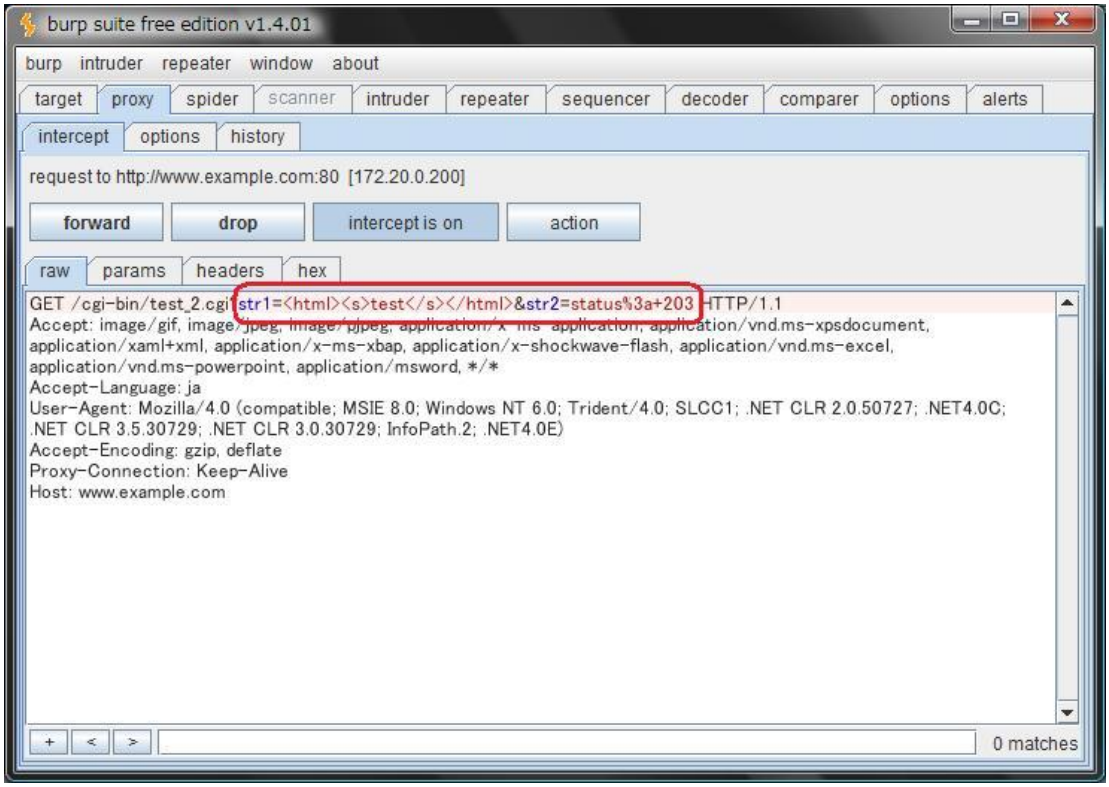

図 2.6-10 : 図 2.6-9のHTTPリクエスト・メッセージ

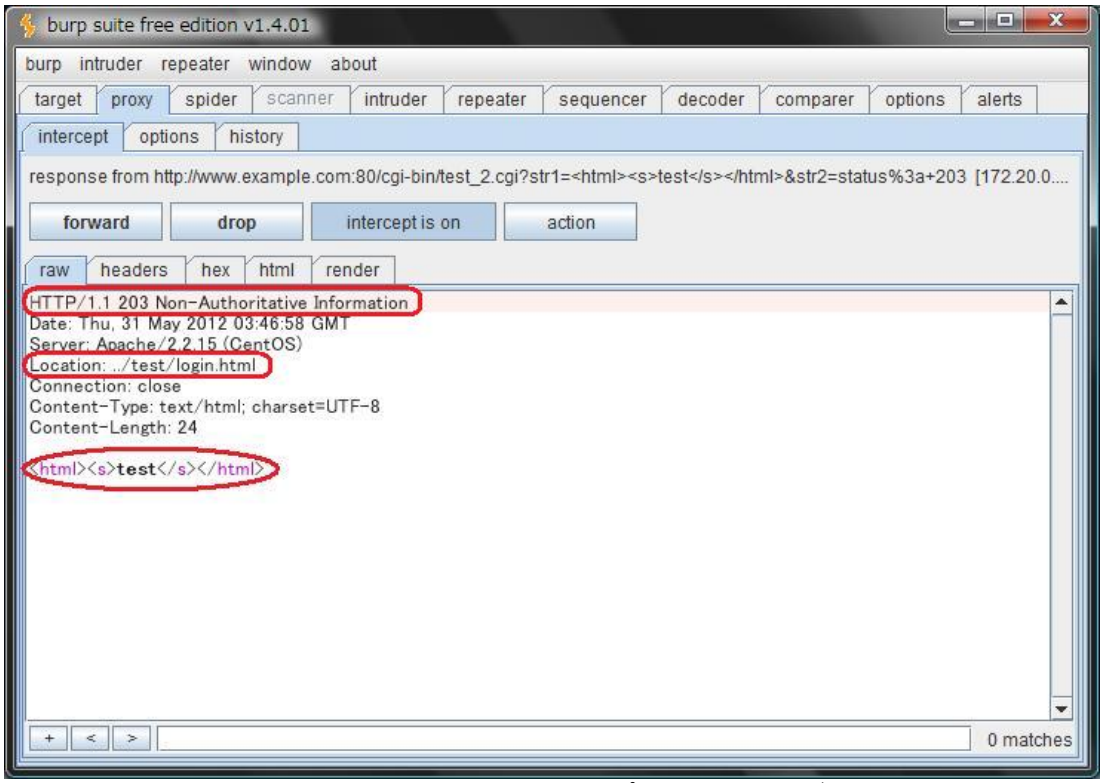

図 2.6-11 : 図 2.6-10のHTTPレスポンス・メッセージ

ボディ部分はXSSが発現する状態になっており、

さらにはHTTP Response Status Codeも改ざんされている

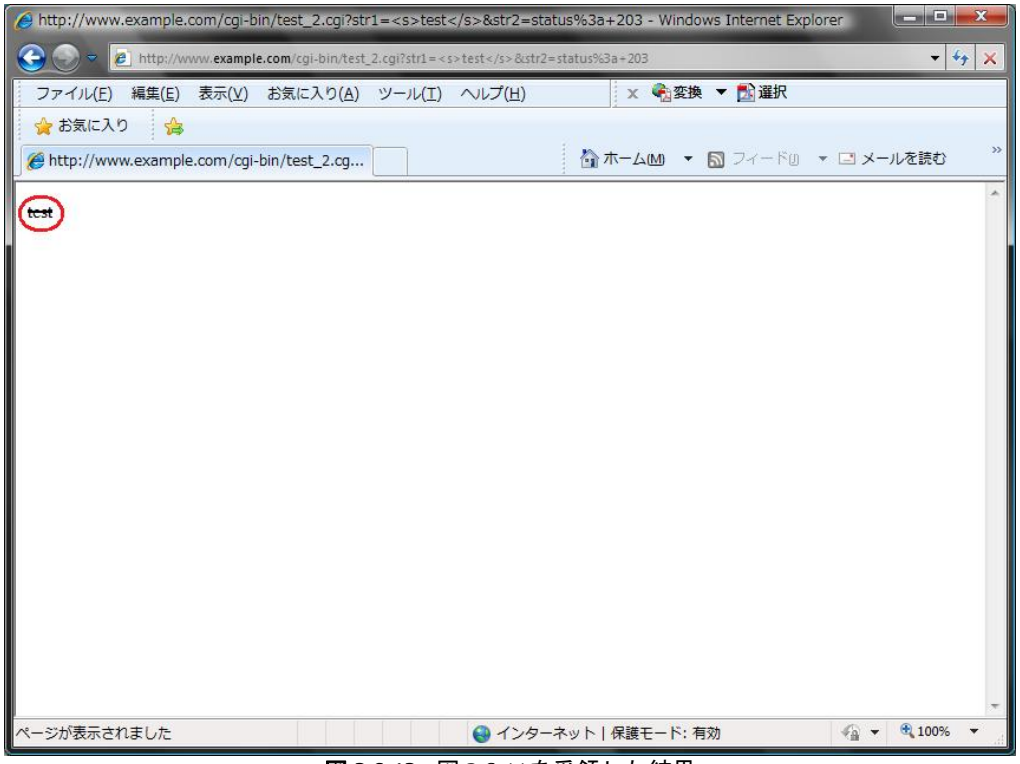

図 2.6-12 : 図 2.6-11を受領した結果

Webブラウザは「login.html」へリダイレクトされず、

HTTPレスポンス・ボディが解釈されXSSが発現する

2.7. 対策

- Status Code によっては、Web ブラウザが無視する HTTP Response の Body であっても、XSS 対策は行っておくこと。
- 不正行為者がヘッダ操作できるような Web アプリケーションを作らないこと。
- 不正行為者がヘッダ操作できるとしても、ヘッダ情報の「値」だけに限定させること。
- ヘッダ情報の操作には、CrLf インジェクション問題がない関数を用いること。

2.8. まとめ (現実的可能性)

本文書では、HTTP Response の Status Code の汚染について検討してきたが、条件として、
- 不正行為者がヘッダ情報の先頭から(または名前の部分から)操作することができる。

という条件が必要であることに留意する必要がある。
これは、Status Code の汚染よりも直接的にヘッダ情報に偽の情報を追記できるということであり、例えば、「Content-Type」ヘッダや「Location」ヘッダなどが追記されるような事態も同時に想定でき、Web ブラウザの種類によっては(Status Code の汚染よりも)深刻なセキュリティ上の問題となるだろう。

また、再度条件を記載するが、
- 不正行為者がヘッダ情報の先頭から(または名前の部分から)操作することができる。

という条件自体が、稀な状況であると思われる。

結論として、稀の状況を想定しないと発現しない現象であると思うが、ヘッダ・インジェクションだけではなく、Status Code が汚染されうるという可能性も考慮する必要がある。

まとめの最後に、実は「Location」ヘッダの出力位置と、Status Code の出力位置、つまり順番によっては、Status Code の汚染が失敗するケースもあり、さらに現実的可能性を小さくするだろう。

3. Status Code=200 以外でのXSS発現可能性と、Webブラウザの設定

3.1. (IE)「HTTPエラーメッセージを簡易表示する」

Microsoft-Internet Explorer には、「HTTP エラーメッセージを簡易表示する」という設定があり、大抵の場合、既定で有効となっている。
この設定が有効な場合、HTTP Response Status Code が「200」以外のエラーなどの場合、HTTP Response Body がコンテンツとして表示されることはないため、XSS を発現させることは困難である。
しかし、無効になっている場合、XSS を引き起こすことが可能となる。

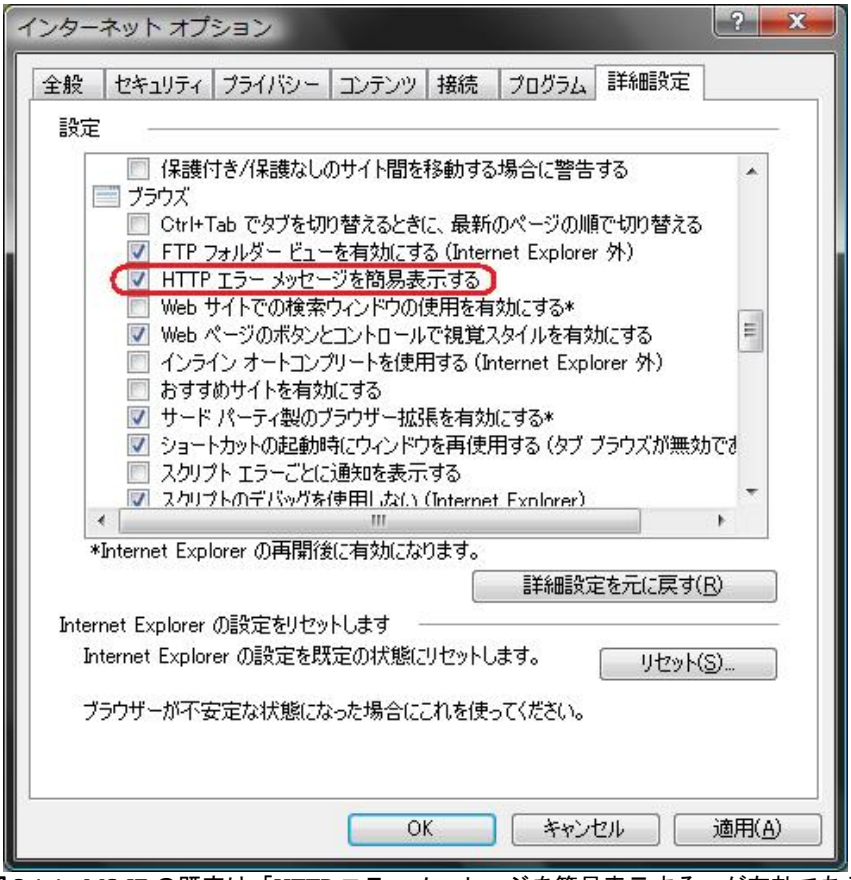

図 3.1-1 : MS-IE の既定は「HTTP エラーメッセージを簡易表示する」が有効である

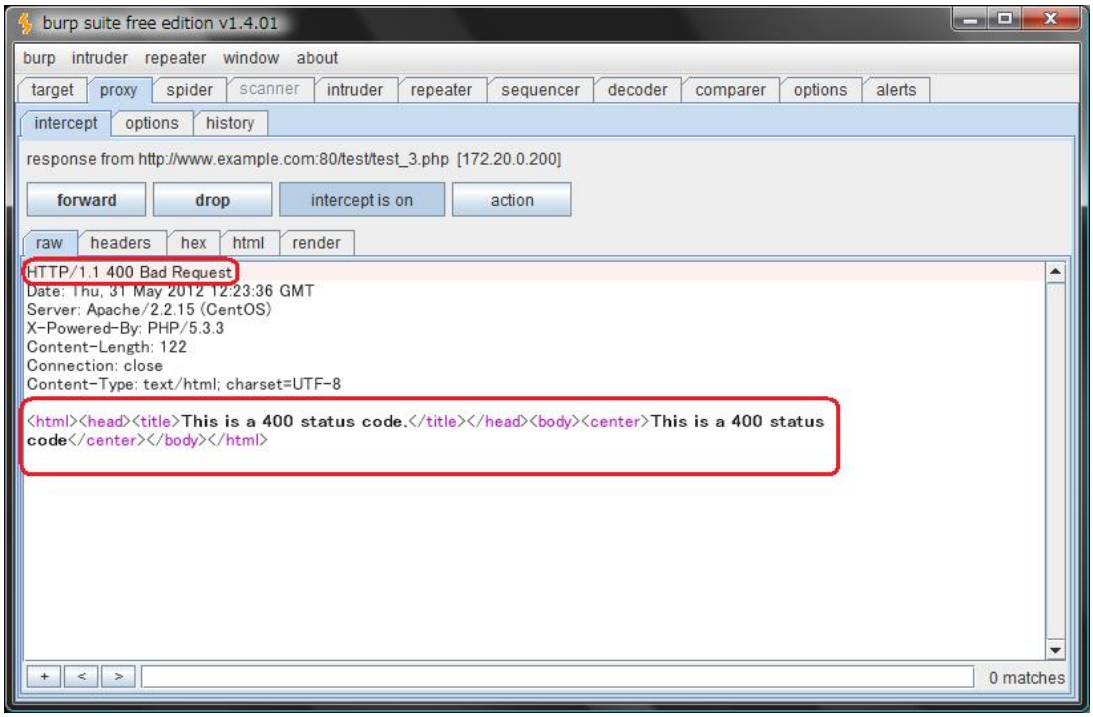

図 3.1-2 : 「HTTP エラーメッセージを簡易表示する」が有効な MS-IE に Status Code が「400」を送る

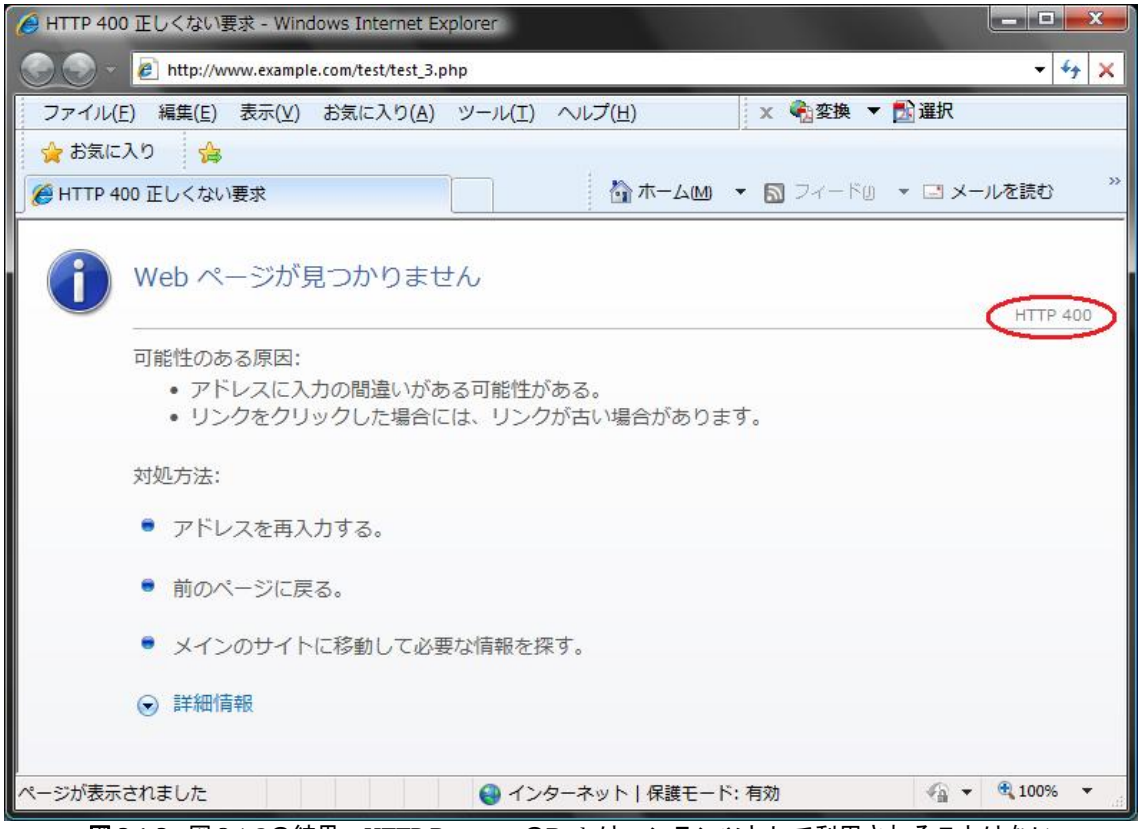

図 3.1-3 : 図 3.1-2の結果。HTTP ResponseのBodyはコンテンツとして利用されることはない

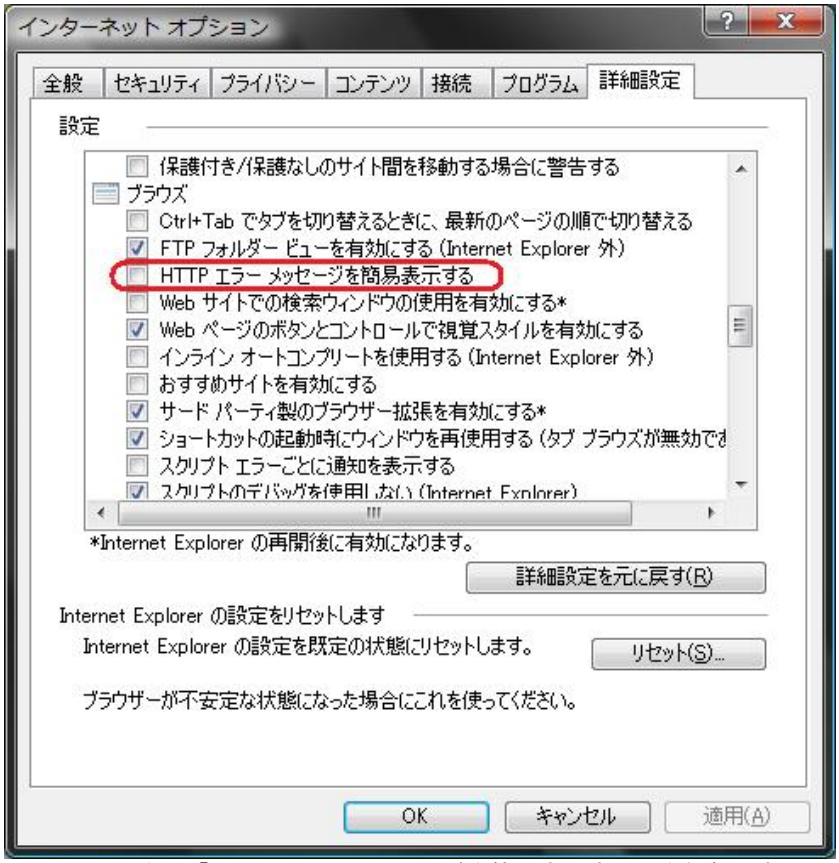

図 3.1-4 : 次に「HTTP エラーメッセージを簡易表示する」を無効にする

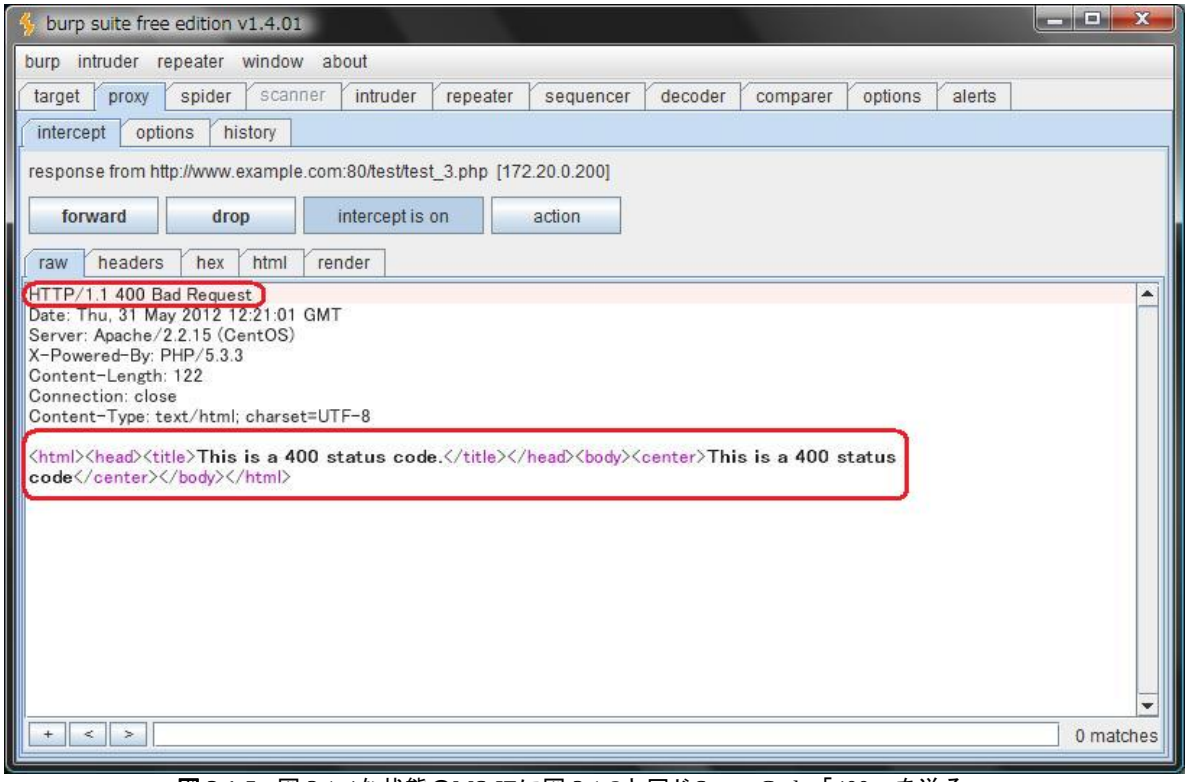

図 3.1-5 : 図 3.1-4な状態のMS-IEに図 3.1-2と同じStatus Code「400」を送る

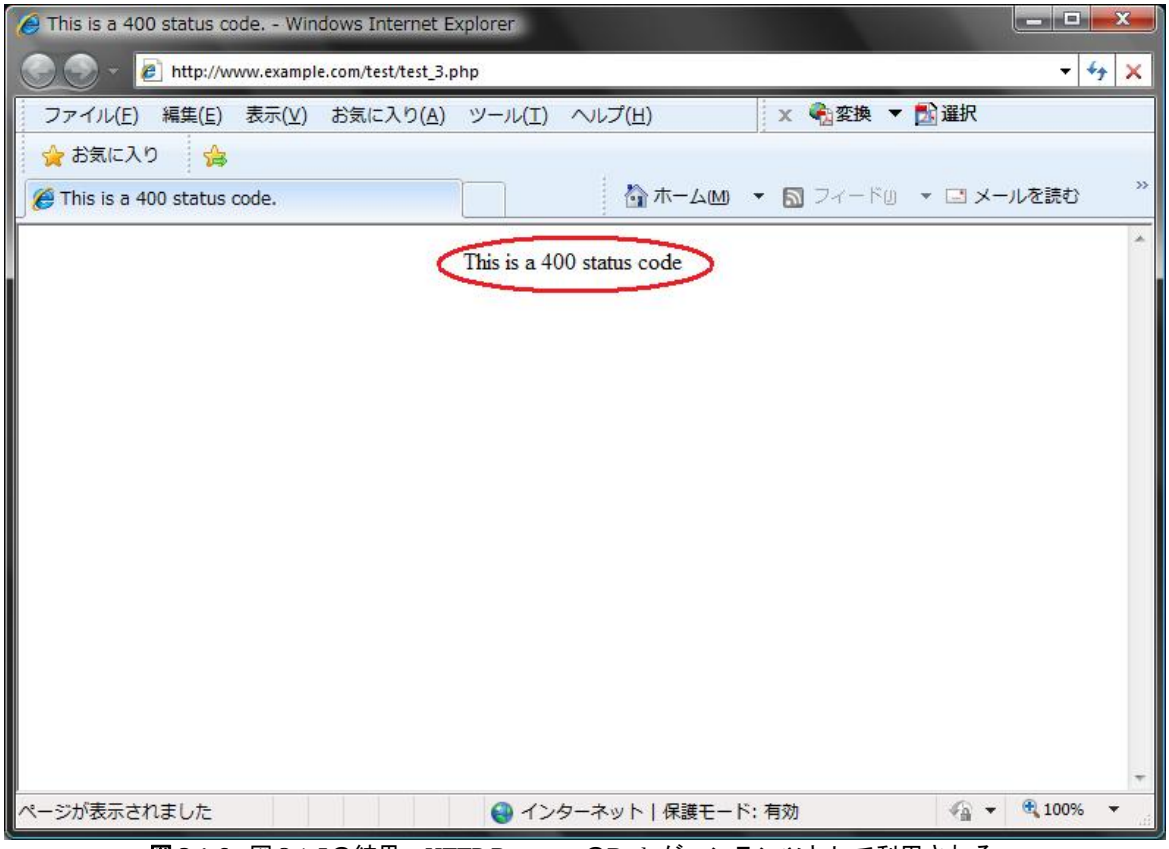

図 3.1-6 : 図 3.1-5の結果。HTTP ResponseのBodyがコンテンツとして利用される

3.2. (Opera)「オートリダイレクトを有効にする」

Opera には、「オートリダイレクトを有効にする」という設定があり、大抵の場合、既定で有効となっている。
この設定が有効な場合、リダイレクト要求を受信した Opera は、自動的にリダイレクトするため、リダイレクト要求の HTTP Response Body はコンテンツとして使われることはなく、XSS を発現させることは困難であるが、無効の場合、自動的にリダイレクトしないため、XSS を発現させることが可能となる。

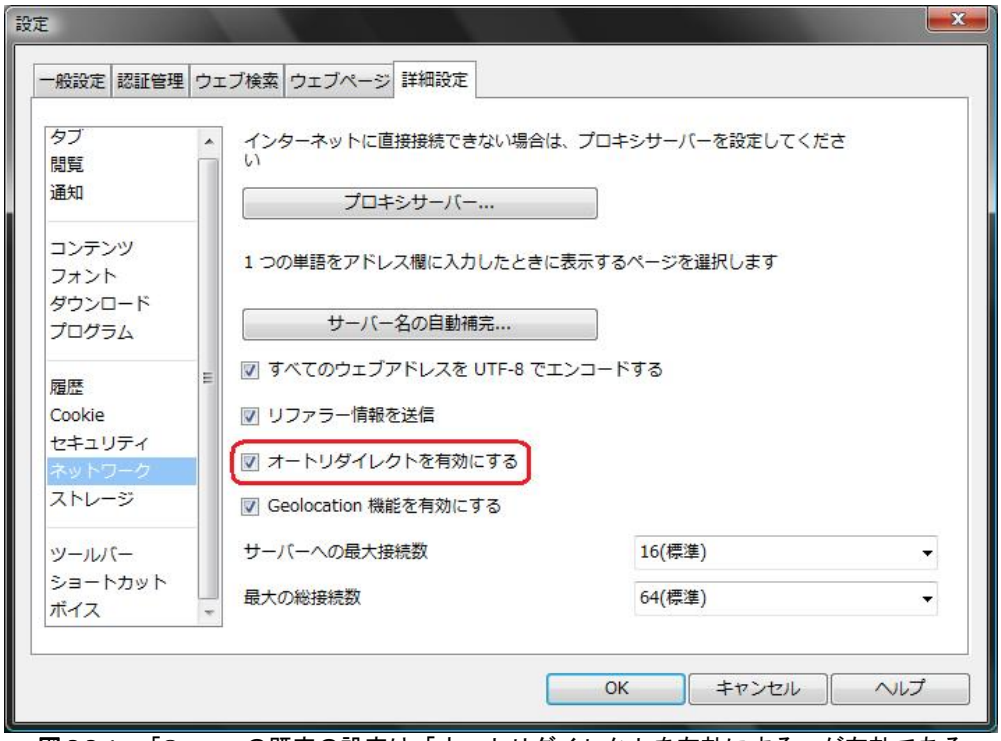

図 3.2-1 : 「Opera」の既定の設定は「オートリダイレクトを有効にする」が有効である

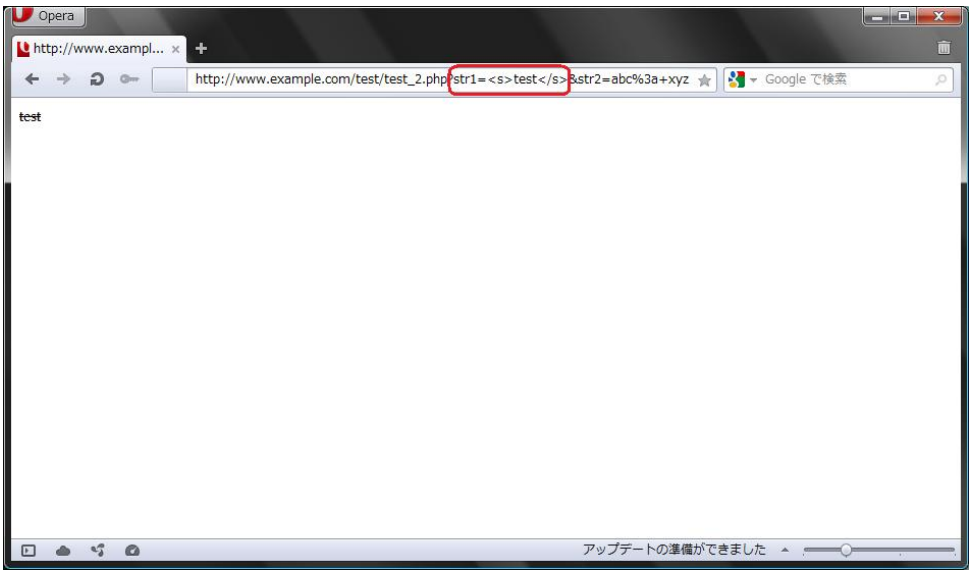

図 3.2-2 : 図 2.3-1に対して変数「str1」に「<」入れてアクセスする

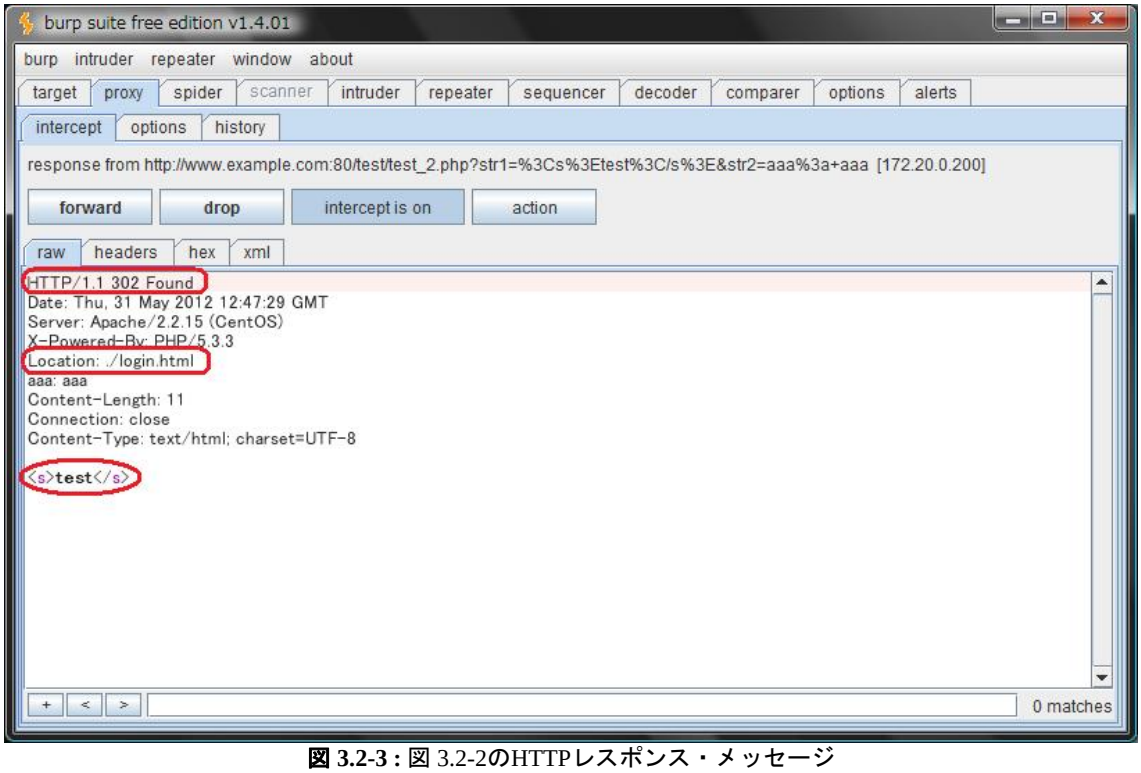

図 3.2-3 : 図 3.2-2のHTTPレスポンス・メッセージ

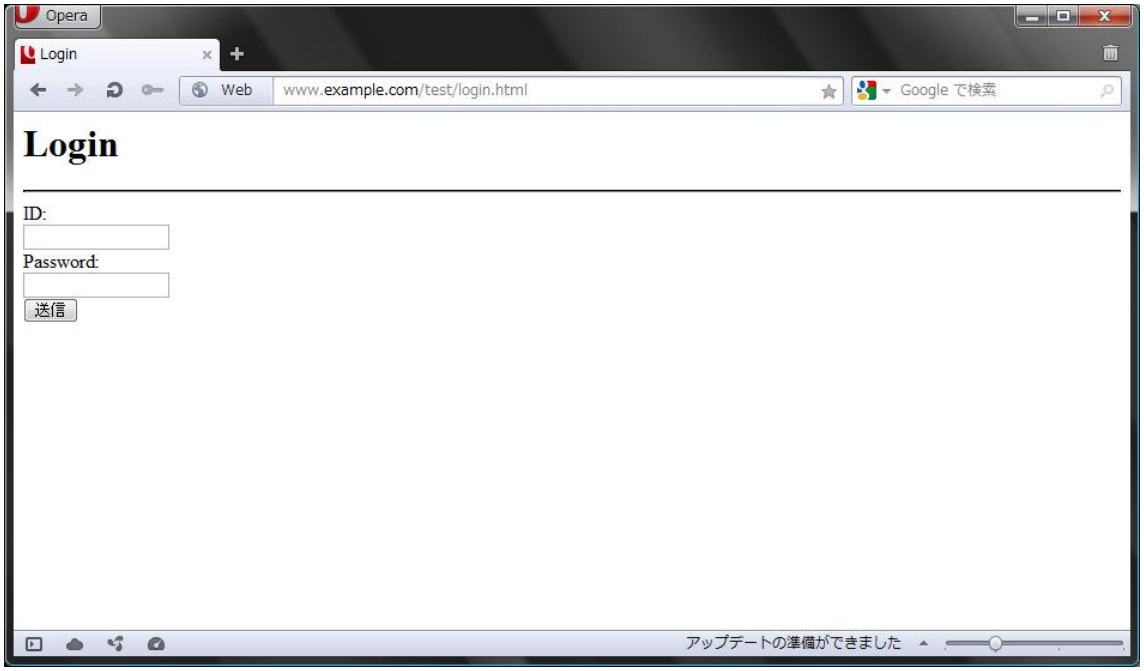

図 3.2-4 : 図 3.2-3の結果、リダイレクトする

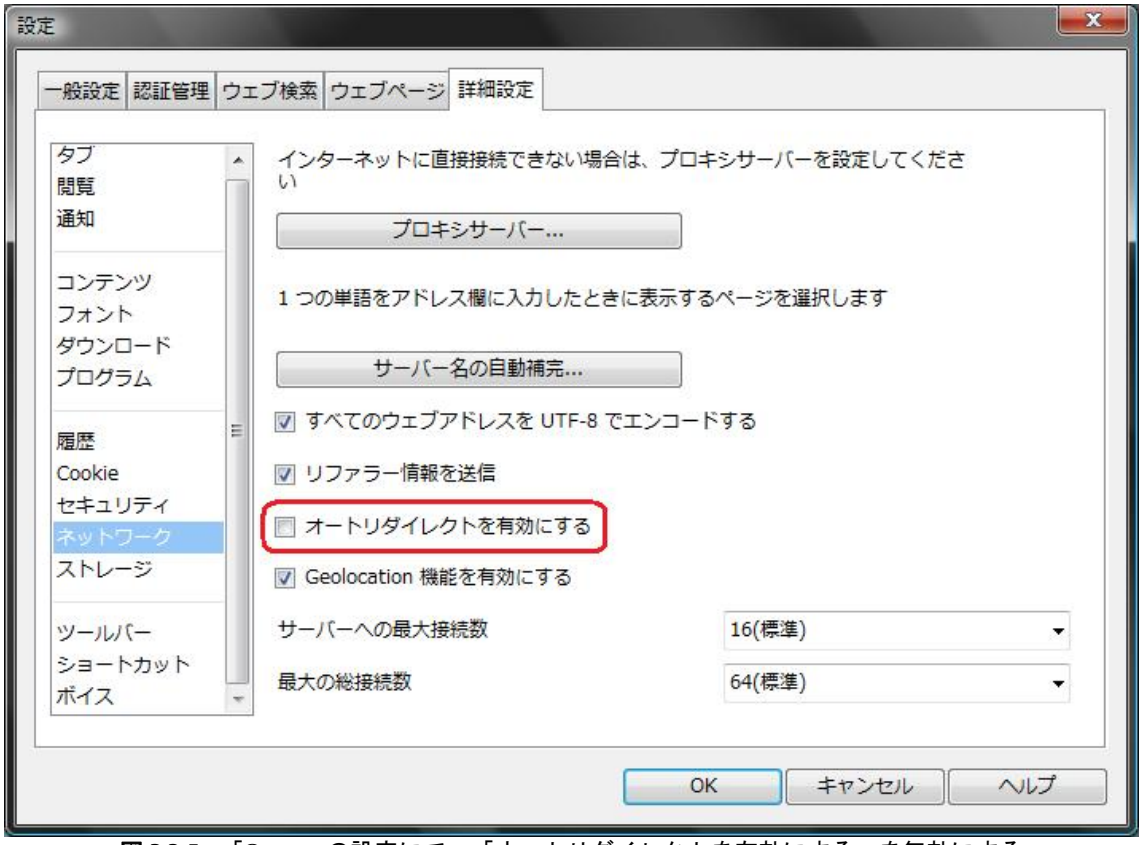

図 3.2-5 : 「Opera」の設定にて、「オートリダイレクトを有効にする」を無効にする

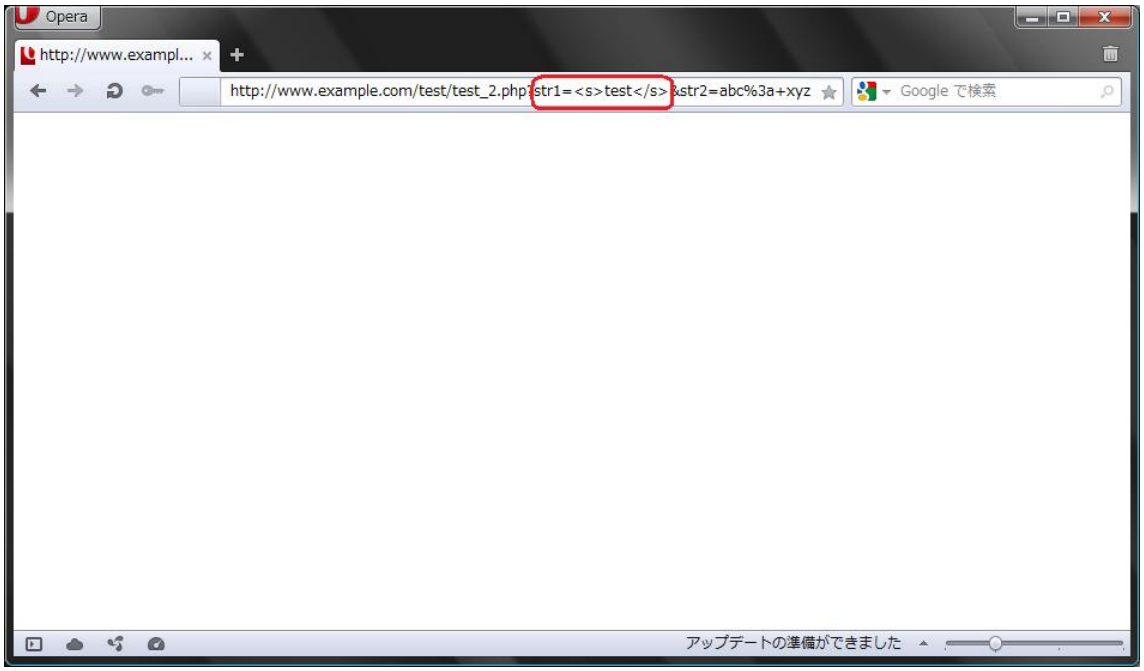

図 3.2-6 : 図 2.3-1に変数「str1」に「<」入れてアクセスする

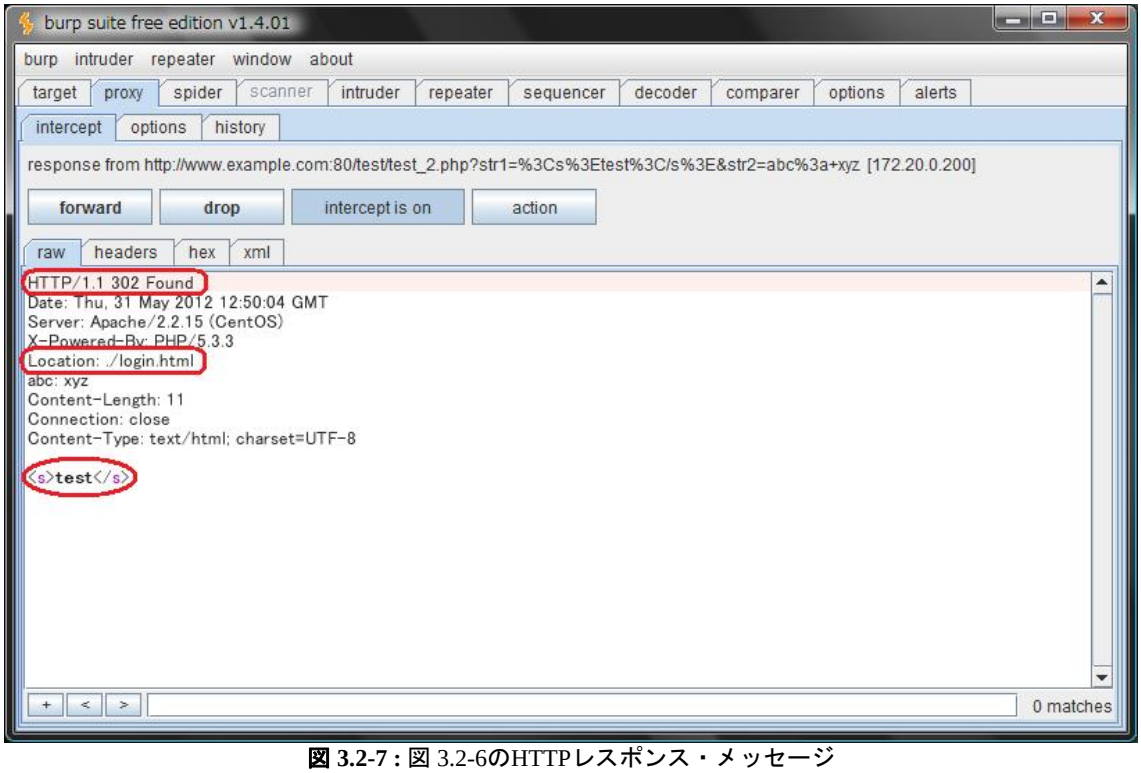

図 3.2-7 : 図 3.2-6のHTTPレスポンス・メッセージ

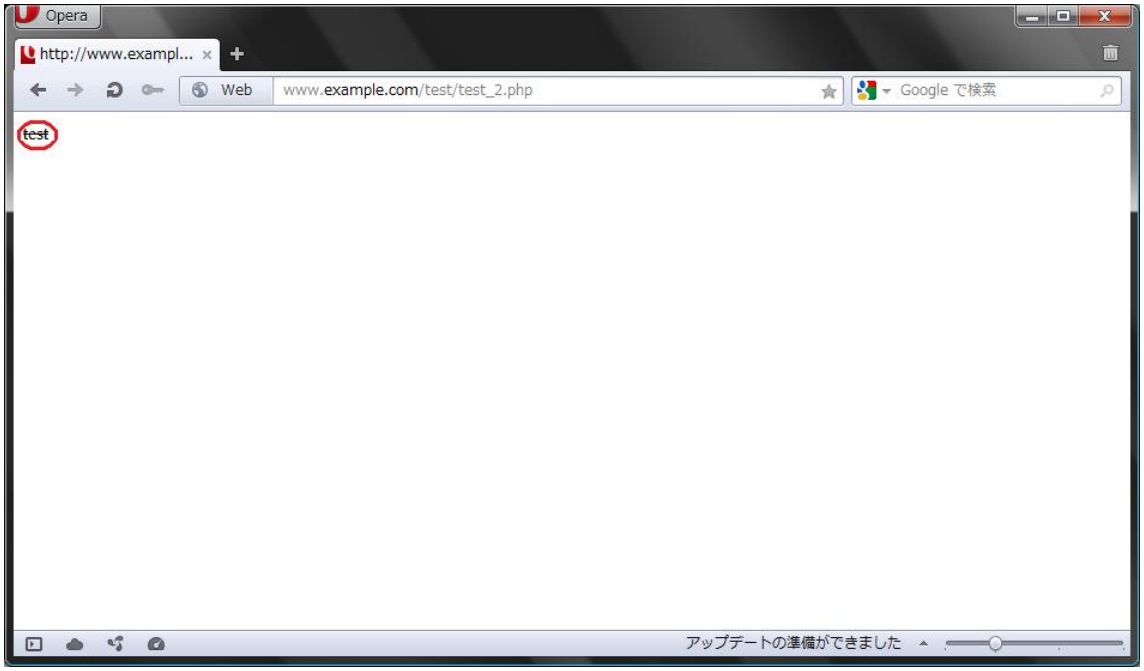

図 3.2-8 : 図 3.2-7の結果、リダイレクトしないため、

HTTP ResponseのStatus Codeを汚染しなくてもXSSが発現する

3.3. (.NET Framework) System.Net.HttpWebRequestクラス

.NET Framework の System.Net.HttpWebRequest クラスには AllowAutoRedirect プロパティがあり、Opera のように、リダイレクトを自動的に行うかどうかを制御することができる。

図 3.3-1 : System.Net.HttpWebRequest#AllowAutoRedirect プロパティ

3.4. まとめ

このように、Web ブラウザの設定によっては、様々なことが考えられるため、このような観点からも Status Code が 200 以外の場合であっても、HTTP Response Body については、XSS 対策(HTML エンコード処理)を念のために行っておくことが、Web アプリケーション開発者には求められるだろう。

4. 検証作業者

NTT コミュニケーションズ株式会社
ソリューションサービス部第四エンジニアリング部門
セキュリティオペレーション担当
本城 敏信
佐名木 智貴

5. 参考

- Security of HTTPHeader ver1.2
  http://rocketeer.dip.jp/secProg/HttpSecurity003.pdf
- PHP Manual - header
  http://php.net/manual/ja/function.header.php
- CGI プログラムの改良案/ヘッダの書き方
  http://hp.vector.co.jp/authors/VA014833/CGI/header.html
- RFC3875: The Common Gateway Interface (CGI) Version 1.1
  http://www.ietf.org/rfc/rfc3875
- System.Net.HttpWebRequest プロパティ
  http://msdn.microsoft.com/ja-jp/library/system.net.httpwebrequest.allowautoredirect(v=vs.95)

6. 履歴

- 2012 年 06 月 15 日 ：ver1.0 最初の公開

7. 最新版の公開URL

http://www.ntt.com/icto/security/data/soc.html

8. 本レポートに関する問合せ先

NTT コミュニケーションズ株式会社
ソリューションサービス部第四エンジニアリング部門
セキュリティオペレーション担当

e-mail: scan@ntt.com

以 上